

AD-A212 063

DTIC FILE COPY

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

POST CRASH FLIGHT ANALYSIS: VISUALIZING FLIGHT RECORDER DATA

by

Mark Jay Christian

June 1989

Thesis Advisor:

Michael J. Zyda

Approved for public release; distribution is unlimited

DTIC
ELECTE
SEP 08 1989
S B D
CB

88

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b. OFFICE SYMBOL (If applicable) Code 37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School		
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000			7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) POST CRASH FLIGHT ANALYSIS: VISUALIZING FLIGHT RECORDER DATA					
12. PERSONAL AUTHOR(S) Christian, Mark J.					
13a. TYPE OF REPORT Master's Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) June 1989	
15. PAGE COUNT 97					
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	3D visual simulation system; flight data recorder visualization		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Previous research has produced a real-time, three dimensional, interactive moving platform simulator (MPS). The simulator utilizes Defense Mapping Agency digital terrain elevation data to generate the three dimensional terrain and runs on Silicon Graphics, Inc. IRIS 4D/70GT graphics workstations. The MPS system has been used as a basis for a variety of military applications. We present here how the MPS system was modified to be utilized as a crash investigation tool for U.S. Army aircraft mishaps. Flight recorder data from the mishap aircraft is used to graphically reconstruct the flight of the aircraft. Flight attitudes, gauge readings, switch positions, warning and advisory light indicators, and flight control inputs are displayed. The visualization of the flight recorder data greatly aids in the analysis of the causes of an aircraft mishap.					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael J. Zyda			22b. TELEPHONE (Include Area Code) (408) 646-2305		22c. OFFICE SYMBOL Code 52

Approved for public release; distribution is unlimited

**Post Crash Flight Analysis:
Visualizing Flight
Recorder Data**

by

Mark Jay Christian
Captain, United States Army
B.S., Iowa State University, 1978

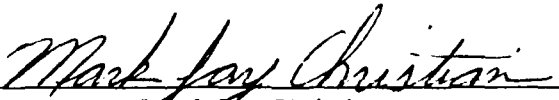
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

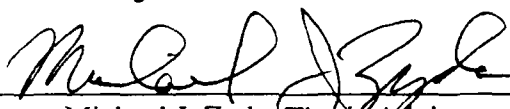
from the

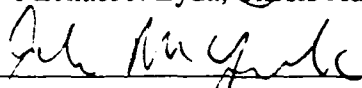
NAVAL POSTGRADUATE SCHOOL
June 1989

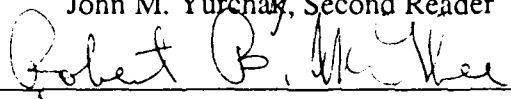
Author:

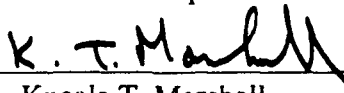

Mark Jay Christian

Approved By:


Michael J. Zyda, Thesis Advisor


John M. Yurchak, Second Reader


Robert B. McGhee, Chairman,
Department of Computer Science


Kneale T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

Previous research has produced a real-time, three dimensional, interactive moving platform simulator (MPS). The simulator utilizes Defense Mapping Agency digital terrain elevation data to generate the three dimensional terrain and runs on Silicon Graphics, Inc. IRIS 4D/70GT graphics workstations. The MPS system has been used as a basis for a variety of military applications. We present here how the MPS system was modified to be utilized as a crash investigation tool for U.S. Army aircraft mishaps. Flight recorder data from the mishap aircraft is used to graphically reconstruct the flight of the aircraft. Flight attitudes, gauge readings, switch positions, warning and advisory light indicators, and flight control inputs are displayed. The visualization of the flight recorder data greatly aids in the analysis of the causes of an aircraft mishap.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	



TABLE OF CONTENTS

I. INTRODUCTION	1
A. ARMY AIRCRAFT ACCIDENT INVESTIGATION.....	1
B. FLIGHT DATA RECORDERS	2
C. GRAPHICAL CRASH INVESTIGATION SYSTEM.....	3
D. LIMITATIONS TO THE PC BASED DISPLAY SYSTEM	3
E. AN IMPROVED GRAPHICAL DISPLAY SYSTEM.....	4
F. ORGANIZATION.....	5
II. FLIGHT RECORDER DATA	6
A. FLIGHT PARAMETERS RECORDED	6
B. DATA INACCURACIES	8
C. CONVERSION OF FLIGHT DATA FILES.....	9
III. SOFTWARE BASIS FOR CAST.....	10
A. FOG-M SIMULATOR	10
B. VEH SIMULATOR	10
C. VEH II SIMULATOR.....	11
D. MOVING PLATFORM SIMULATOR (MPS).....	11
IV. MODIFICATION OF MPS FOR THE IMPLEMENTATION OF CAST	12
A. DELETION OF CODE FROM MPS	12
B. MODIFICATION OF DATA STRUCTURES	14
C. GRAPHICS WINDOW LAYOUT MODIFICATIONS.....	19
D. MODELING AND DISPLAY OF A HELICOPTER	24
1. MPS Platform Display Programs.....	24

2. CAST Helicopter Display Programs.....	27
E. USER POPUP MENU INTERFACE MODIFICATIONS	29
1. Popup Menus Deleted	29
2. Popup Menu Options Deleted.....	30
3. Popup Menus Added.....	31
F. USER CONTROL INTERFACE MODIFICATIONS.....	32
1. Mouse Control Modifications	32
2. Dial Box Control Modifications	34
3. Graphics Displayed In The INDWIN Graphics Window	44
G. PLATFORM POSITION UPDATE MODIFICATIONS.....	44
1. Position Update In MPS	44
2. Position Update In CAST	45
H. MODIFICATION OF VIEWING PERSPECTIVE.....	46
I. ALTITUDE SMOOTHING	48
1. Altitude Smoothing Algorithms.....	48
2. Algorithm Testing.....	49
3. Additional Altitude Smoothing.....	50
V. CONCLUSIONS AND RECOMMENDATIONS	52
A. SYSTEM PERFORMANCE.....	52
B. CAST SYSTEM LIMITATIONS AND FUTURE RESEARCH.....	53
1. Aircraft Placement And UTM Grid Coordinates.....	53
2. Operational Area Boundaries.....	54
3. Modeling Of Additional Aircraft	54
4. X-Y Plotting Of Flight Data	55
5. Selective Frame Display	55
6. Adverse Flight Condition Warning.....	55

C. CONCLUSIONS	55
APPENDIX A CAST USER'S GUIDE	57
A. PREPROCESSING FLIGHT DATA FILES	57
B. STARTING THE CAST SYSTEM	58
C. SELECTING AN AREA OF OPERATION.....	58
D. ENTERING AIRCRAFT FOR DISPLAY	59
E. DELETING AIRCRAFT FROM THE SYSTEM.....	59
F. SELECTING A PLATFORM TO OPERATE	60
G. CHANGING THE EYEPOINT	60
H. SELECTING AN ALTITUDE OPTION.....	61
I. SELECTING A FRAME DELAY OPTION.....	61
J. EXITING CAST.....	62
APPENDIX B SELECTED CAST SOURCE CODE	63
LIST OF REFERENCES	85
INITIAL DISTRIBUTION LIST	87

LIST OF TABLES

TABLE 1	LOTUS 1-2-3 OUTPUT FORMAT.....	6
TABLE 2	DISCRETE FIELDS.....	7
TABLE 3	PARAMETER SAMPLE RATES AND UNITS OF MEASUREMENT.....	8
TABLE 4	MPS MODULES DELETED IN THE CAST IMPLEMENTATION.....	13
TABLE 5	PARAMETER FREQUENCY DECODING	17
TABLE 6	ALTITUDE SMOOTHING ALGORITHMS.....	49
TABLE 7	SMOOTHING ALGORITHM RESULTS	50

LIST OF FIGURES

Figure 1	MPS Vehicle Record Structure.....	14
Figure 2	CAST Vehicle Record Structure.....	15
Figure 3	Flight Data Record Structure	16
Figure 4	Vehicle and Flt_data_rec Linked Lists	18
Figure 5	MPS Graphics Window Layout.....	20
Figure 6	CAST Graphics Window Layout.....	21
Figure 7	CAST Graphics Windows.....	23
Figure 8	Display of the COBRA Helicopter Model.....	25
Figure 9	MPS Polygon Drawing Code.....	26
Figure 10	CAST Polygon Drawing Code	28
Figure 11	MPS Mouse Controls.....	33
Figure 12	CAST Mouse Controls.....	35
Figure 13	MPS Dial Box Controls for Driving a Vehicle.....	36
Figure 14	MPS Dial Box Controls for Flying a Missile	37
Figure 15	CAST Dial Box Controls With Eyepoint Inside Aircraft	39
Figure 16	CAST Dial Box Controls With Eyepoint Outside Aircraft	40
Figure 17	View From Inside a COBRA Helicopter	42
Figure 18	View of Three COBRA Helicopters in Formation Flight.....	43
Figure 19	Display of the Horizon in CAST and MPS.....	47
Figure 20	Altitude Smoothing.....	51

ACKNOWLEDGEMENT

I would like to thank Major Bill Teter, USA, for the use of his viewing transformation routines. The incorporation of these routines into the CAST program made the realistic depiction of aerial flight possible.

I. INTRODUCTION

A. ARMY AIRCRAFT ACCIDENT INVESTIGATION

U.S. Army aircraft mishaps are classified as Class A, B, C, D, or E accidents based upon the extent of damage to the aircraft involved and the severity of injury or loss of life of personnel involved [Ref. 1:p. 13]. Class A accidents are the most severe mishaps and Class E accidents are the least severe mishaps. The accident rate for U.S. Army aircraft has shown a steady decrease over the past few years. The Class A accident rate has decreased from a rate of 2.94 accidents per 100,000 hours flown during Fiscal Year 1985 to a rate of 1.84 accidents per 100,000 hours flown during Fiscal Year 1988. During the same time, the Class A to Class C accident rate dropped from 8.75 accidents per 100,000 hours flown to 4.76 accidents per 100,000 hours flown [Ref. 2:p. 35]. Much of this decrease can be attributed to the increased emphasis on flight safety and aircraft maintenance within the Army's Aviation commands as well as the professionalism displayed by Army aviators. The findings produced from investigations of accidents that have occurred have also contributed to the decrease in the accident rate. These findings have identified defective components in Army aircraft, verified flight environments that are not suitable for Army aircraft, and found limitations in flight systems [Ref. 3:p. 7-1]. For these investigation findings to be effective, the data obtained must be timely, complete, and properly analyzed [Ref. 4:p. 2-1].

The process of investigating an aircraft crash is complicated. Although Army aviators are trained to be highly observant, during the high pressure of an emergency

situation they can not be expected to have observed everything which would like to be known by the crash investigation team. Indeed, many times it is the case that something not observed led to an accident. The crash investigation team must rely upon the accounts of the mishap given by the pilots of the aircraft. Unfortunately, at times there are no survivors of a crash and the crash investigation team has only eyewitness accounts of the crash (if any) and the wreckage upon which to base their findings.

B. FLIGHT DATA RECORDERS

In March 1986, the Army Chief of Staff directed that a flight data recorder project be implemented to aid in the crash investigation process. Currently, a flight data recorder has been developed for the UH-60 Black Hawk helicopter. The recorder has been installed in approximately two hundred high-risk mission Black Hawk helicopters at Fort Rucker, Alabama; Fort Bragg, North Carolina; and Fort Campbell, Kentucky. This is an analog recorder with a continuous tape for recording of the flight data. Two hours of flight data can be recorded before the tape is overwritten with new data. A new digital flight data recorder is currently being developed to replace the analog recorder. A flight data recorder for the AH-64 Apache is also presently under development. The goal is to eventually equip all Army aircraft with flight data recorders [Ref. 5:p. 36].

The UH-60 flight data recorder records sixty four flight and advisory parameters [Ref. 6]. These parameters describe the flight attitude in terms of altitude, airspeed, roll, pitch, and yaw as well as the readings from instrument panel gauges, positions of switches, and status of flight advisory lights. The sample rate for each parameter varies depending upon the usefulness of the parameter. Sample rates range from

eight samples per second for flight attitude parameters to one sample per second for switch position and flight advisory lights. An elapsed time counter is recorded with each set of samples. The aircraft tail number and an error check flag are also recorded.

C. GRAPHICAL CRASH INVESTIGATION SYSTEM

To assist in the crash investigation process, a personal computer (PC) based graphics display system has been developed under a government contract for the United States Army Safety Center (USASC) at Fort Rucker, Alabama. The system is run on a Compaq 386/20 Model 130 computer. This system utilizes the data obtained from the flight data recorder of an aircraft involved in a crash. The system has the following capabilities:

1. Any eight of the sixty four data parameters can be plotted against time at once. The system user can specify the time interval to be used for the plot.
2. All Army aircraft have been modeled and can be displayed on the output screen. The aircraft can be "flown" using the data from the flight recorder. The eye position for watching the aircraft's flight can be placed inside the aircraft to provide a pilot's perspective or can be placed anywhere outside the aircraft to observe the total aircraft as it is "flown". As with the graphical plotting of parameters, the time interval for the flight can be specified.

D. LIMITATIONS TO THE PC BASED DISPLAY SYSTEM

The Compaq PC based graphical display system has several limitations. These limitations are caused by the limited amount of memory in the computer and the relatively small size of the display screen. The system is also not fully developed and at present lacks features which would enhance its usefulness. Limitations of the current system include:

1. The model "flying" is done over flat ground which is not representative of the flight environment in which the accident occurred.

2. The plotting of flight parameters and model "flying" can not be displayed simultaneously.
3. The field of view of the pilot cannot be changed (as when the pilot is wearing night vision goggles).
4. Only a single aircraft can be displayed at one time.

E. AN IMPROVED GRAPHICAL DISPLAY SYSTEM

In the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School, we have developed an improved graphical display system for crash investigation. The name of the system is the Crash Analysis Simulation Tool (CAST) and it is run on a Silicon Graphics, Inc. IRIS 4D/70GT graphics workstation. The system programming is based upon the Moving Platform Simulator (MPS) which was developed by past thesis students in the Graphics and Video Laboratory [Ref. 7]. The new crash investigation system provides a real-time, interactive, three dimensional depiction of the flight under investigation. The limitations of the PC based system are eliminated in the workstation based system. The workstation system has the following capabilities:

1. Simultaneous display of flight data parameters and the flight of the aircraft. Flight data parameters are displayed both graphically and numerically.
2. Simultaneous display of multiple aircraft in flight.
3. Placement of the eye point at either the pilot or copilot stations with a variable field of view or outside of the aircraft.
4. Flight over three dimensional terrain. The system uses a Defense Mapping Agency (DMA) Digital Terrain Elevation Database (DTED) to construct the three dimensional terrain. By obtaining the terrain database for the crash area, the system can "fly" the aircraft over the terrain on which the crash occurred.

It is envisioned that the PC based version of the crash investigation system will be used for immediate, preliminary analysis of an aircraft mishap. The PC based

system is portable and can be transported to the crash site. The workstation version will then be utilized for follow-on, detailed analysis of the crash after leaving the crash site.

F. ORGANIZATION

The sections of this chapter have presented the motivation for the development of an enhanced graphical display system for the visualization of flight recorder data. Chapter II describes the format of the flight recorder data and its conversion to the format used in CAST. Chapter III discusses the software basis for the CAST program. Chapter IV details the modification of the Moving Platform Simulator (MPS) program for the implementation of CAST. Chapter V lists limitations in the CAST system and gives recommendations for further research and development of CAST. Appendix A contains a user's guide for CAST. Appendix B contains source listings for new programs which were written to implement CAST.

II. FLIGHT RECORDER DATA

A. FLIGHT PARAMETERS RECORDED

When an aircraft is involved in a mishap, the flight data recorder is recovered and the data is extracted by a crash investigation team from the United States Army Safety Center. The team uses a device which transcribes the raw flight data into a Lotus 1-2-3 format which is used by the Compaq PC based graphical display system. The transcription device utilizes sixty seven of the seventy two parameters recorded by the flight data recorder. A line counter is added to each set of parameters output by the transcription device. Table 1 lists the Lotus 1-2-3 output format of the data transcription device with the minimum and maximum values for each parameter [Ref. 8]. Table 2 lists the fifty discrete parameters recorded. Table 3 lists the sample rate and unit of measurement for each parameter.

TABLE 1 Lotus 1-2-3 OUTPUT FORMAT

FIELD	DESCRIPTION	FORMAT	MIN VALUE	MAX VALUE
1	Line Counter	I6	0	+999999
2	Combined Time	F9.3	0.000	+99999.999
3	Pitch Attitude	F6.1	-82.0	+82.0
4	Roll Attitude	F6.1	-180.0	+180.0
5	Yaw Attitude	F5.1	0.0	+360.0
6	Longitudinal Stick	F5.1	-5.0	+105.0
7	Latitudinal Stick	F5.1	-5.0	+105.0
8	Pedal Position	F5.1	-5.0	+105.0
9	Collective Stick	F5.1	-5.0	+105.0
10	Airspeed	F5.1	0.0	+180.0
11	Altitude	I5	-1000	+20000
12	Altitude Rate	I5	-6000	+6000
13	Engine #1 Torque	F5.1	0.0	+142.0
14	Engine #2 Torque	F5.1	0.0	+142.00
15	Rotor RPM	F5.1	0.0	+130.0
16	Stabilator Position	F5.1	-10.0	+40.0
17-66	Discretes	I1	0	+1
67	Tail Number	I3	0	+999
68	Check Word Error	I1	0	+1

TABLE 2 DISCRETE FIELDS

FIELD	DESCRIPTION
17	Master Warning
18	Fire Master Warning
19	Stabilator
20	Stabilator Auto Reset
21	Stabilator Slew Up
22	Stabilator Slew Down
23	SAS Off
24	#1 Generator Caution
25	#2 Generator Caution
26	#1 Converter Caution
27	#2 Converter Caution
28	AC Essential Bus Caution
29	DC Essential Bus Caution
30	#1 Primary Servo Caution
31	#2 Primary Servo Caution
32	#1 Hydraulic Pump Caution
33	#2 Hydraulic Pump Caution
34	Tail Rotor Quadrant Control
35	#1 Tail Rotor Servo Caution
36	#2 Tail Rotor Servo Caution
37	Boost Servo Off Caution
38	Trim Failure
39	FPS
40	APU Failure
41	APU On
42	APU Generator On
43	#1 Reservoir
44	#2 Reservoir
45	Backup Reservoir
46	Backup Pump On
47	#1 Engine Out
48	#2 Engine Out
49	Low Rotor RPM
50	Drag Beam
51	#1 Tail Rotor Shutoff Valve
52	#2 Tail Rotor Shutoff Valve
53	Boost Shut Off
54	SAS/Pitch Boost Shut Off
55	Pitch Trim System
56	Pilot Assist Shut Off
57	#1 Primary Servo Shut Off
58	#2 Primary Servo Shut Off
59	FPS Switch
60	Primary 1st Stage Servo Switch
61	Primary 2nd Stage Servo Switch
62	SAS #1 Switch
63	SAS #2 Switch
64	Trim Switch
65	Tail Rotor Servo Switch
66	Pitch Trim Turnoff Valve

**TABLE 3 PARAMETER SAMPLE RATES
AND UNITS OF MEASUREMENT**

FIELD	DESCRIPTION	SAMPLE RATE	UNIT OF MEASUREMENT
1	Line Counter	8/sec	-----
2	Combined Time	8/sec	Seconds
3	Pitch Attitude	8/sec	Degrees
4	Roll Attitude	8/sec	Degrees
5	Yaw Attitude	8/sec	Degrees
6	Longitudinal Stick	4/sec	Percent
7	Latitudinal Stick	4/sec	Percent
8	Pedal Position	4/sec	Percent
9	Collective Stick	4/sec	Percent
10	Airspeed	2/sec	Knots
11	Altitude	2/sec	Feet
12	Altitude Rate	2/sec	Feet/Minute
13	Engine #1 Torque	2/sec	Percent
14	Engine #2 Torque	2/sec	Percent
15	Rotor RPM	2/sec	Percent
16	Stabilator Position	1/sec	Degrees
17-66	Discretes	1/sec	On/Off
67	Tail Number	1/sec	-----
68	Check Word Error	1/sec	True/False

B. DATA INACCURACIES

Inaccurate data is inevitable in most flight data files. The Commander of the United States Army Safety Center has directed that no modifications be made to the data obtained from flight recorders. As such, unusual graphics are occasionally displayed by the CAST program. By analyzing the data displayed with the graphics, it is possible to determine when inaccurate data is being used to produce the graphics.

The airspeed indicator, altimeter, and vertical speed indicator all operate off of the pitot-static system of an aircraft [Ref. 5:p. 2-17 - 2-31]. The airspeed indicator in many cases is inaccurate below thirty knots of forward airspeed and on most aircraft can not indicate sideward or rearward flight speeds. Most flight speeds below thirty knots forward airspeed will be recorded as zero airspeed by the flight data recorder. The altimeter and vertical speed indicators are subject to atmospheric

pressure errors and can also be affected by a helicopter's rotor wash. Instrument lag is also prevalent in the vertical speed indicator unless an instantaneous vertical speed indicator is installed in the aircraft.

C. CONVERSION OF FLIGHT DATA FILES

Several Lotus 1-2-3 format flight data files were obtained on floppy disk from the United States Army Safety Center. These data files were converted to a new format which is used by the CAST program. A utility program was written in the C programming language to accomplish the data conversion. This program can be found in Appendix B.

The CAST format data files are more readable than the Lotus 1-2-3 format data files. The Lotus 1-2-3 format files have no spacing between parameter fields. The CAST format data files provide spacing between parameter fields to increase their readability.

The fifty discrete parameters are written as a string of ones and zeros in the Lotus 1-2-3 format data files. The fifty discrete values are converted into two unsigned integers in the CAST format data files. These integers serve as bit fields to determine which discrete parameters are active (on). Twenty five bits in each integer are used for discrete parameters with seven bits unused in each.

III. SOFTWARE BASIS FOR CAST

The CAST system is based upon the Moving Platform Simulator (MPS) which was completed in November 1988 by thesis students working in the Graphics and Video Laboratory at the Naval Postgraduate School. The MPS system consolidated and upgraded vehicle simulators which had been developed by previous thesis students.

A. FOG-M SIMULATOR

The initial development of a vehicle simulation system was completed in June 1987. This system was a Fiber Optically Guided-Missile simulator which was developed for the United States Army Combat Developments Experimentation Center at Fort Ord, California [Ref. 10:p. 11-12]. This system was run on a Silicon Graphics, Inc. IRIS 3120 graphics workstation. The system allowed an operator to fly the FOG-M over three dimensional terrain generated from digital terrain elevation data of the Fort Hunter-Liggett, California area. Vehicles traversed the terrain on preset courses. The operator was able to target, track, and destroy these vehicles with the FOG-M.

B. VEH SIMULATOR

The FOG-M simulator was upgraded to the Vehicle (VEH) simulator. This work was completed in December 1987 [Ref 11:p. 8]. Like the FOG-M simulator, VEH was run on the IRIS 3120 graphics workstation. The VEH simulator improved the user interface of the FOG-M system. In VEH, the FOG-M control panel was simulated so that the operator could control the missile's speed, heading, altitude,

and camera position. A driver's view was also provided for the ground vehicles. A more efficient terrain drawing algorithm, based upon the vehicle driver's or missile camera's field of view was also implemented in VEH [Ref. 11:p. 9-10].

C. VEH II SIMULATOR

The VEH simulator was then modified to be run on more powerful Silicon Graphics, Inc. workstations. VEH was first modified to be run on the IRIS 4D/70G graphics workstation. This modification took advantage of the higher level graphics capabilities of the 4D/70G. Popup menus were implemented and the system was operated under the MEX window management system [Ref. 7:p. 5].

Further modifications were made to VEH II to allow it to be run on the IRIS 4D/70GT graphics workstation. The system was now operated under the 4Sight window management system [Ref. 7:p. 5].

D. MOVING PLATFORM SIMULATOR (MPS)

MPS provided further modifications to VEH II to utilize the built-in graphics hardware of the IRIS 4D/70GT. MPS added capabilities such as realistic, variable light intensities based upon the month and time of day, Z-buffering for hidden surface removal, and RGB color mode. An improved terrain drawing algorithm was developed which uses distance attenuation of the terrain surface to improve the simulator performance. A broadcast network capability was added to allow communications between multiple simulators over an Ethernet link. [Ref. 7:p. 8]

IV. MODIFICATION OF MPS FOR THE IMPLEMENTATION OF CAST

The MPS program provided a good basis for the development of CAST. The three dimensional terrain and two dimensional map drawing modules of MPS were suitable for usage in a crash analysis simulator. However, extensive modifications had to be made to MPS for the implementation of CAST. Many of the operations available in MPS were not appropriate for aircraft crash analysis. The coding for these features was removed from MPS. Many new software modules were added to MPS to provide the capabilities required for crash analysis. The resulting CAST program is a scaled-down version of MPS with crash analysis specific capabilities.

A. DELETION OF CODE FROM MPS

Being a moving platform simulator, MPS can display a variety of ground and airborne platforms. The platforms displayed by MPS include:

1. Open and Covered Jeeps
2. Trucks
3. Tanks
4. Fiber Optically Guided Missiles

Since Army aviation accidents very rarely involve ground vehicles, it was decided to remove all ground vehicles for the CAST simulator. The MPS program modules which generated the ground vehicles were deleted. Modules which provided the user interface for the control of the ground vehicles and which calculated the vehicle driver's look position were also removed.

The FOG-M also was not needed for the CAST system. The display routines for the FOG-M were deleted and were replaced with routines to draw a helicopter. The program modules which provided the FOG-M's tracking capability of ground vehicles were removed.

CAST is intended to be utilized as a stand-alone crash investigation tool. Therefore, the MPS networking capability was deemed unnecessary for CAST. The ability to simultaneously simulate the flights of multiple aircraft and to interactively change the cockpit of reference for the display of the aircraft was found to be satisfactory for a proper analysis of an aircraft mishap. Thus, the networking code was removed from MPS for the CAST implementation.

Table 4 lists the MPS modules which were deleted for the CAST implementation.

TABLE 4 MPS MODULES DELETED IN THE CAST IMPLMENTATION

.C MODULES DELETED		
add_network_veh.c	check_for_packets.c	display_indbox.c
display_slider.c	display_tracked_message.c	do_change_speed.c
do_driving_menu.c	do_the_defaults.c	drawflame.c
drawintank.c	drawjeep.c	drawopenjeep.c
drawroller.c	drawtank.c	drawtire.c
drawtrack.c	drawtruck.c	drawwreck.c
event_driving.c	flamenormals.c	handle_tracking.c
initveh.c	intanknormals.c	jeepnormals.c
limit_cursor_pick.c	network.c	network_receive.c
openjeepnormals.c	reset_tilt.c	rollernormals.c
semaphore.c	setcontrols.c	slowturn.c
tanknormals.c	tirenormals.c	tot_num_ground_veh.c
tracking_check.c	tracknormals.c	trucknormals.c
update_look_pos.c	update_net_veh_pos.c	
.H MODULES DELETED		
Flamedata.h	Intankdata.h	Jeepdata.h
Network.h	Openjeepdata.h	Rollerdata.h
Tankdata.h	Tiredata.h	Trackdata.h
Truckdata.h		

B. MODIFICATION OF DATA STRUCTURES

The MPS program utilizes a record structure to maintain information on each platform being displayed. The records are stored as a linked list. Networking, tracking, course, speed, translation, rotation, terrain position, and screen location data are maintained for each platform. Figure 1 shows the MPS record structure used to maintain this data.

```
typedef struct vehicle {
    int      net_id;    /* Platform ID number for networking purposes */
    short    pick_id;   /* Pick ID number for targeting purposes */
    short    t;         /* Platform type */
    Coord    x;         /* X translation */
    Coord    y;         /* Y translation */
    Coord    z;         /* Z translation */
    short    tilt;      /* X rotation */
    float    ang;       /* Y rotation */
    short    inc;       /* Z rotation */
    short    gridx;     /* X grid index to draw platform in */
    short    gridz;     /* Z grid index to draw platform in */
    float    vel;       /* Velocity in meters per second */
    float    alt;       /* Altitude if it is a FOG-M missile */
    float    cse;       /* Compass course */
    float    sx;        /* X screen coord for icon on contour map */
    float    sy;        /* Y screen coord for icon on contour map */
    Boolean   track_flag; /* If type is a ground platform then */
                                /* FALSE = not being tracked */
                                /* TRUE = is being tracked */
                                /* If type is a FOG-M missile */
                                /* FALSE = not currently tracking */
                                /* TRUE = is tracking */
    struct vehicle *track; /* If type is a ground platform then */
                                /* It is a pointer to the FOG-M, otherwise */
                                /* It points to the ground platform */
    struct vehicle *next; /* Next node in the list */
} Vehicle;
```

Figure 1 MPS Vehicle Record Structure

The MPS Vehicle record structure was modified for CAST by first eliminating fields which were utilized for networking and tracking. Fields were then added to store the pilot's viewing directions and the tail number of the aircraft. Pointer fields were added to access the aircraft's flight recorder data. Fields were added to store the Universal Transverse Mercator (UTM) grid coordinates of the aircraft as it traverses over the terrain. The UTM grid coordinate fields are not used in the current version of CAST, but they were added for future development of the system. Figure 2 shows the CAST record structure for maintaining data on each aircraft.

```
typedef struct vehicle {
    int    tail_nbr;    /* Aircraft tail number */
    short  t;           /* Platform type */
    Coord  x;           /* X translation */
    Coord  y;           /* Y translation */
    Coord  z;           /* Z translation */
    float  roll;        /* X rotation (in degrees) */
    float  ang;         /* Y rotation (in radians) */
    float  pitch;       /* Z rotation (in degrees) */
    short  gridx;       /* X grid index to draw platform in */
    short  gridz;       /* Z grid index to draw platform in */
    float  vel;         /* Velocity in meters per second */
    float  alt;         /* Altitude (in feet) */
    float  n_hi;        /* UTM northing high byte (future use) */
    float  n_lo;        /* UTM northing low byte (future use) */
    float  e_hi;        /* UTM easting high byte (future use) */
    float  e_lo;        /* UTM easting low byte (future use) */
    float  cse;         /* Compass course (in degrees) */
    float  lookaz;      /* Viewer's look dir side to side (deg's) */
    float  lookel;      /* Viewer's look dir up and down (deg's) */
    float  sx;          /* X screen coord for icon on contour map */
    float  sy;          /* Y screen coord for icon on contour map */
    Boolean end_of_flight_data; /* Indicates all flight data read */
    Flt_data_rec *flt_data; /* Pointer to flight data record list */
    Flt_data_rec *curr_flight_data_rec; /* Current record being used */
    struct vehicle *next; /* Next node in the vehicle list */
} Vehicle;
```

Figure 2 CAST Vehicle Record Structure

A record structure was added to the CAST program to store the flight recorder data. These records are linked together to form a doubly linked list. Both forward and backward links were required to implement an option which allows the user to manually step forward and backward through the flight data. Figure 3 shows the record structure for the flight data.

```
typedef struct f1t_data_rec {
    int frequency; /* freq rate for data record (8/4/2/1) */
    int frame_no; /* Frame (output line) counter */
    float time; /* Combined time */
    float pitch; /* Pitch attitude (degrees) */
    float roll; /* Roll attitude (degrees) */
    float yaw; /* Yaw attitude (degrees heading) */
    float long_cyclic; /* Longitudinal stick (%) */
    float lat_cyclic; /* Latitudinal stick (%) */
    float pedals; /* Pedal position (%) */
    float collective; /* Collective position (%) */
    float airspeed; /* Airspeed (knots) */
    int altitude; /* Altitude (feet MSL) */
    float s_alt; /* Smoothed altitude (feet MSL) */
    int alt_rate; /* Change in altitude (feet/minute) */
    float eng1_torque; /* Engine #1 torque (%) */
    float eng2_torque; /* Engine #2 torque (%) */
    float rotor_rpm; /* Main rotor rpm (%) */
    float stab_pos; /* Stabilator position (degrees) */
    unsigned int discrete1; /* First 25 discretes (1/0) */
    unsigned int discrete2; /* Second 25 discretes (1/0) */
    struct f1t_data_rec *next; /* Next flight data record */
    struct f1t_data_rec *prev; /* Previous flight data record */
} F1t_data_rec;
```

Figure 3 Flight Data Record Structure

When an aircraft is added to the list of vehicles to display, the flight recorder data is read from a data file specified by the user. Each frame of flight recorder data is stored in a separate flight data record. Because flight parameters are recorded at varying intervals, the utilization rate for the record fields varies in a cyclic pattern.

The frequency field of the flight data record indicates which parameters are stored in the record. The decoding for the frequency field is shown in Table 5.

TABLE 5 PARAMETER FREQUENCY DECODING

FREQUENCY VALUE	# FLIGHT DATA FIELDS USED	DESCRIPTION OF PARAMETERS RECORDED
8	5	Parameters recorded at a rate of eight samples per second (5 total).
4	9	Parameters recorded at a rate of eight samples per second (5 total) plus the parameters recorded at a rate of four samples per second (4 total).
2	15	Parameters recorded at a rate of eight samples per second (5 total) plus the parameters recorded at a rate of four samples per second (4 total) plus the parameters recorded at a rate of two samples per second (6 total).
1	18	All parameters are recorded as described above plus selected parameters recorded at a rate of one sample per second (3 total). This includes the stabilator position and the fifty discrete parameters combined as unsigned integers.

One linked list is maintained which contains all of the **Vehicle** records which represent the aircraft being displayed. Each **Vehicle** record points to its own linked list of **Flt_data_rec** records which contain the flight recorder data for the aircraft. The relationship between the **Vehicle** and **Flt_data_rec** linked lists is shown in Figure 4.

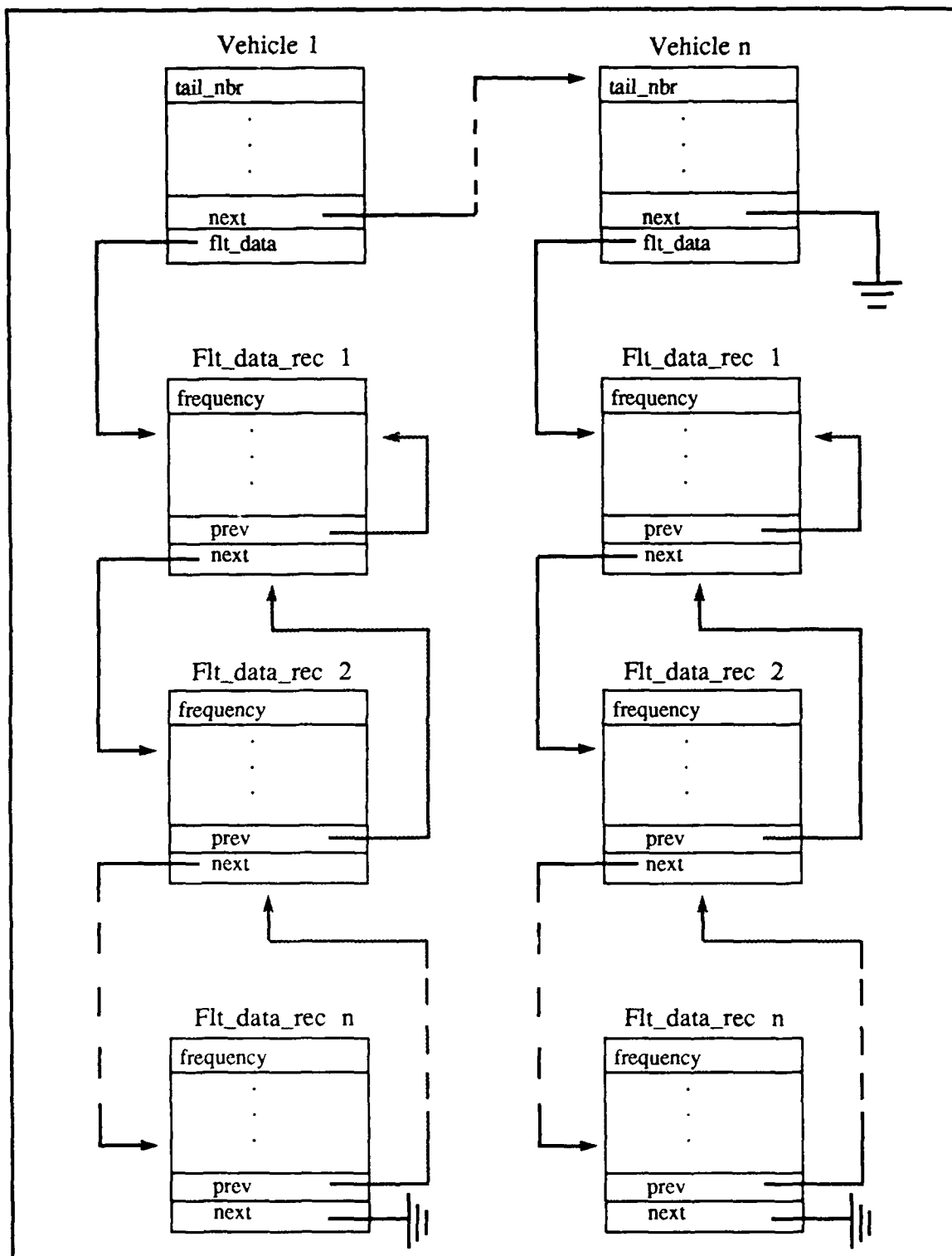


Figure 4 Vehicle and Flt_data_rec Linked Lists

C. GRAPHICS WINDOW LAYOUT MODIFICATIONS

The MPS system operates under the 4Sight window management system. Four graphics windows are utilized in MPS for the display of data and the animation of the moving platforms. The windows are arranged so that they appear as a single window on the display screen. Figure 5 depicts the layout of the windows in MPS. The MPS windows are utilized as follows:

1. MAPWIN - This is a dual purpose window used to display both two and three dimensional graphics. A two dimensional contour map produced from the digital terrain elevation data is displayed which enables the user to select a 10km X 10km area of operation and to position platforms within the area of operation. The three dimensional animation of the platforms moving throughout the area of operation is also displayed in this window.
2. NAVWIN - This window is used to display a two dimensional contour map of the area of operation. Indicated on this map are the positions of the platforms within the area of operation and the direction of travel and field of view of the platform currently under the operator's control.
3. INDWIN - This window displays mouse and dial box controls utilized for the user interface with the system. There are two control displays; one for the control of ground platforms and one for the control of a FOG-M. Numerical data is displayed in this window for the platform under operator control. The speed and heading of the platform is displayed as is the altitude if the platform is a FOG-M. The operator's viewing direction and field of view data are also displayed. Target distance data is displayed for a FOG-M which is tracking a ground target.
4. MENUWIN - This window is used to provide performance, sunlight, and networking data to the user. The number of polygons currently drawn in the MAPWIN and the current and average number of frames per second being displayed in the MAPWIN are shown to provide performance statistics to the user. The user selected month and time of day are displayed along with sunrise, midday, and sunset data for the month selected. A message is also displayed which indicates whether networking is enabled or disabled.

The 4Sight window management system allows resizing of windows. The MPS system takes advantage of this capability by providing a user menu option to resize the windows. All windows are resized when this option is selected so that they maintain the appearance of a single window. The resize option places a lower bound

on the overall combined size of the four windows (600 X 480 pixels) so that the user can not accidentally resize the windows too small for use.

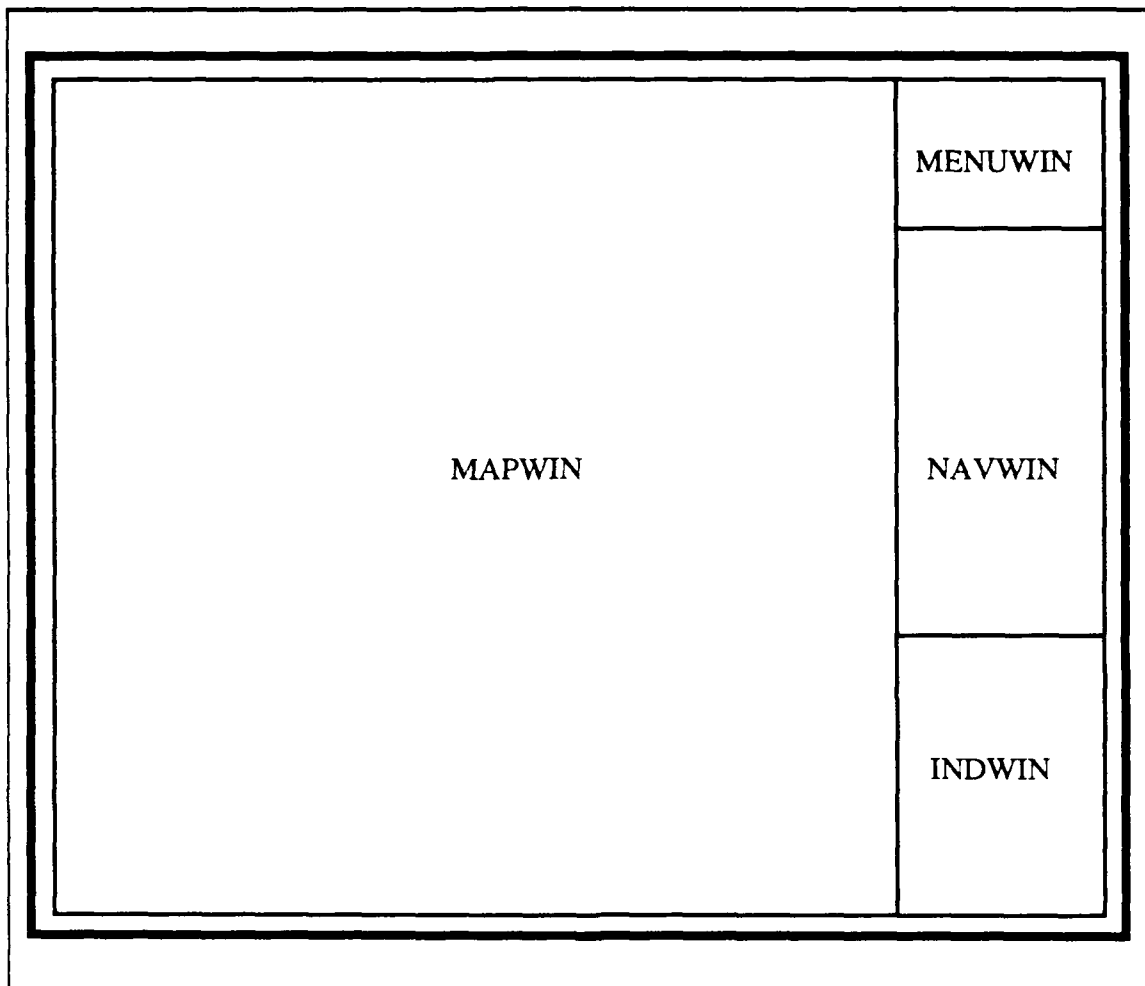


Figure 5 MPS Graphics Window Layout

All of the windows used in MPS were appropriate for the CAST system, however additional windows were needed to graphically display the flight recorder data. Two new windows were added in the CAST system. The MAPWIN was decreased in size, but still retained its square dimensions. One of the new windows

was placed above the MAPWIN and one to the right side of the MAPWIN. Figure 6 depicts the layout of the graphics windows in the CAST system.

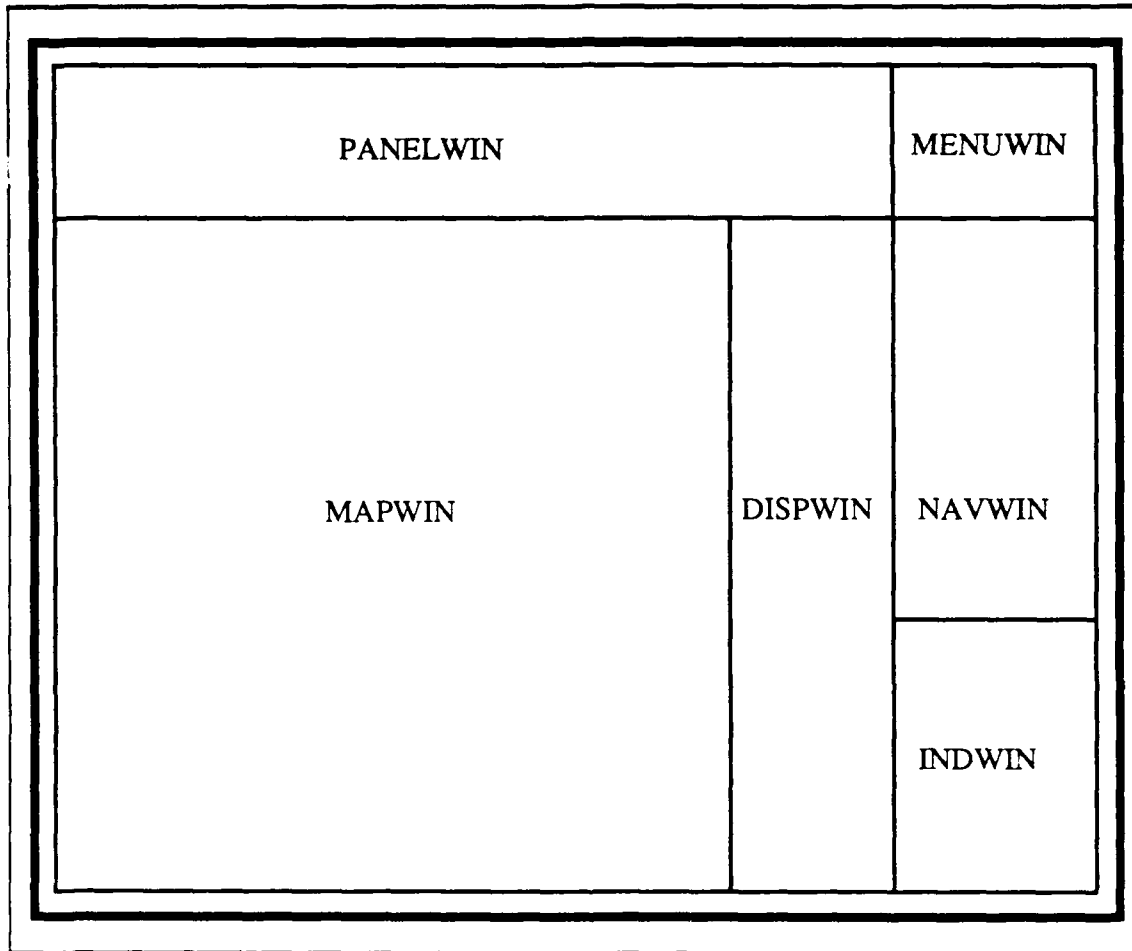


Figure 6 CAST Graphics Window Layout

The windows in the CAST system have the following utilizations:

1. MAPWIN - This window has the same usage in the CAST system as in the MPS system.
2. NAVWIN - This window also provides the same graphical display in the CAST system as in the MPS system.

3. INDWIN - As in the MPS system, this window displays the mouse and dial box controls for the user interface. Unlike MPS, there is only one control display in CAST. Selected flight data is also displayed in this window. Section F of this chapter contains a detailed description of the user interface and data displayed in this window.

4. MENUWIN - This window provides essentially the same information to the user in the CAST system as in the MPS system. Since the networking capability has been removed from CAST, the networking message displayed in this window in MPS has been replaced with the tail number of the aircraft currently under operator control in CAST.

5. PANELWIN - This window is utilized to display the status of the fifty discrete parameters recorded in the flight data. Five rows of ten rectangular boxes are displayed in the window. Each box is labeled with a discrete parameter name. When a discrete parameter is inactive (off), the box for the parameter is displayed in the window background color (cyan). When a parameter becomes active (on), the corresponding parameter box is colored yellow. The overlay planes of this window are used to provide an input area for entering a flight recorder data file name when an aircraft is being added for display.

6. DISPWIN - This window is used to graphically display the positions of the aircraft's cyclic, collective, and anti-torque pedal controls as recorded in the flight data. The longitudinal and latitudinal cyclic stick percentages and the degrees of pitch and roll of the aircraft are displayed under the cyclic graphic. The percent of collective and vertical speed data are displayed under the collective graphic and the percent of left pedal is displayed under the pedals graphic. The stabilator position is also graphically displayed as are the percent of rotor rpm and the percent of power utilization for the aircraft engines.

Figure 7 shows the CAST system windows with typical graphics displayed.

The CAST system retains the menu option provided in MPS for the resizing of the windows. The resizing procedure is the same as in MPS with all windows being resized together to maintain the appearance of a single window. The same lower bound is placed on the size of the combined windows as is used in MPS.

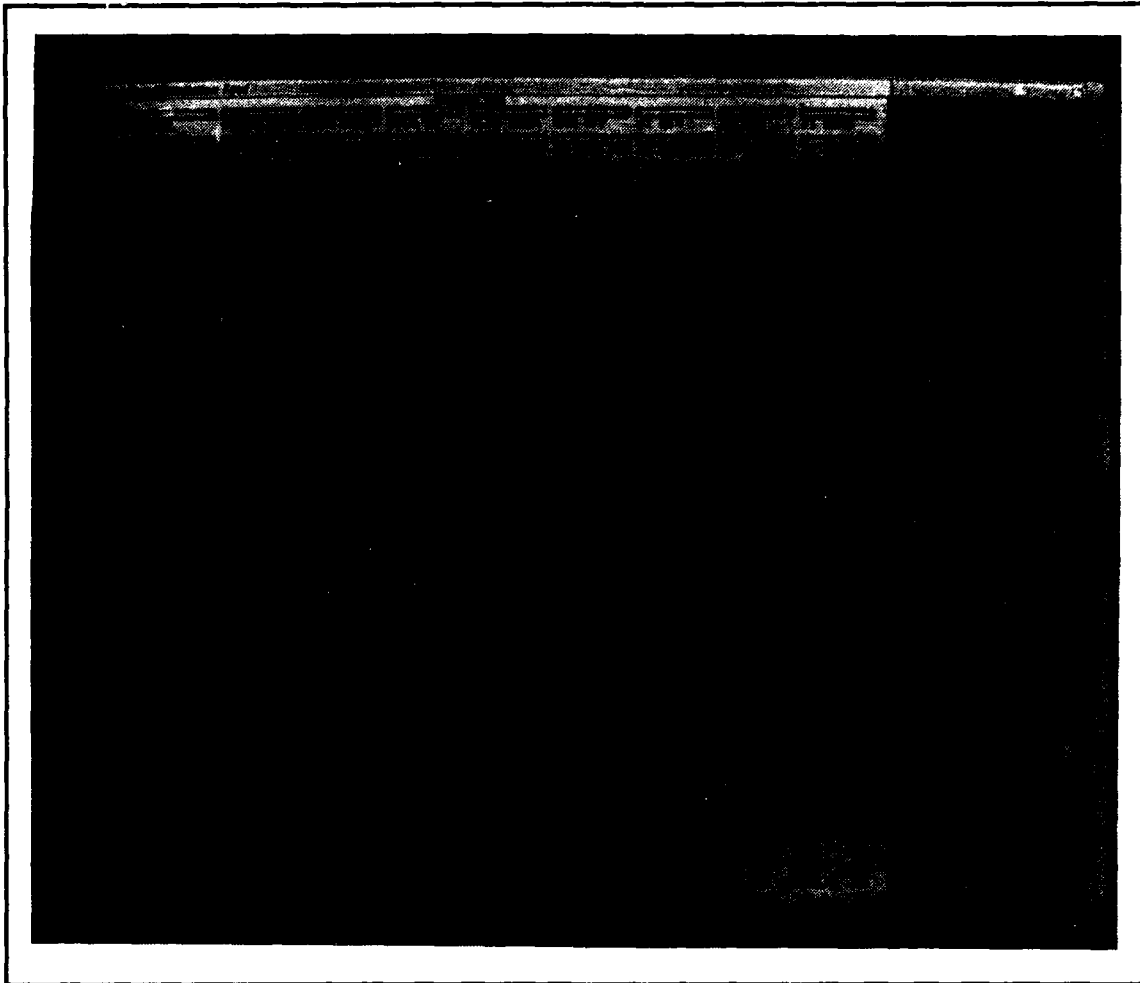


Figure 7 CAST Graphics Windows

D. MODELING AND DISPLAY OF A HELICOPTER

Since the only airborne platform modeled in MPS is a missile, a helicopter model was needed for the development of CAST. For the initial implementation of CAST it was unimportant as to which helicopter was modeled. What was important was to display a realistic representation of a helicopter with a minimum number of polygons. As the number of polygons drawn directly affects system performance, the goal was to model a helicopter with 200 or less polygons.

An AH-1 (COBRA) was chosen as the helicopter to model for the initial CAST implementation. This helicopter was chosen because the angular shape of the airframe was far easier to model than the rounded airframes of other helicopters like the UH-60 BLACK HAWK. Cylindrical components of the COBRA such as the skid tubes, engine exhaust pipe, and infrared suppressor were modeled as four or eight-sided cylinders. The COBRA was modeled with a total of 185 polygons which provided enough detail for a very realistic looking helicopter and met the goal of 200 or less polygons. Figure 8 shows the COBRA as it is displayed by the CAST system.

1. MPS Platform Display Programs

Each polygon used to construct a platform in the MPS system is stored in a two dimensional array of floating point numbers. The first dimension of the array corresponds to the number of vertices in the polygon. The second dimension is always three for the X, Y, and Z world coordinates of each vertex. A vector of three floating point numbers is associated with each polygon. The vector is used to store the normal of the polygon. The normal is utilized with the lighting routines during the display of the platform.

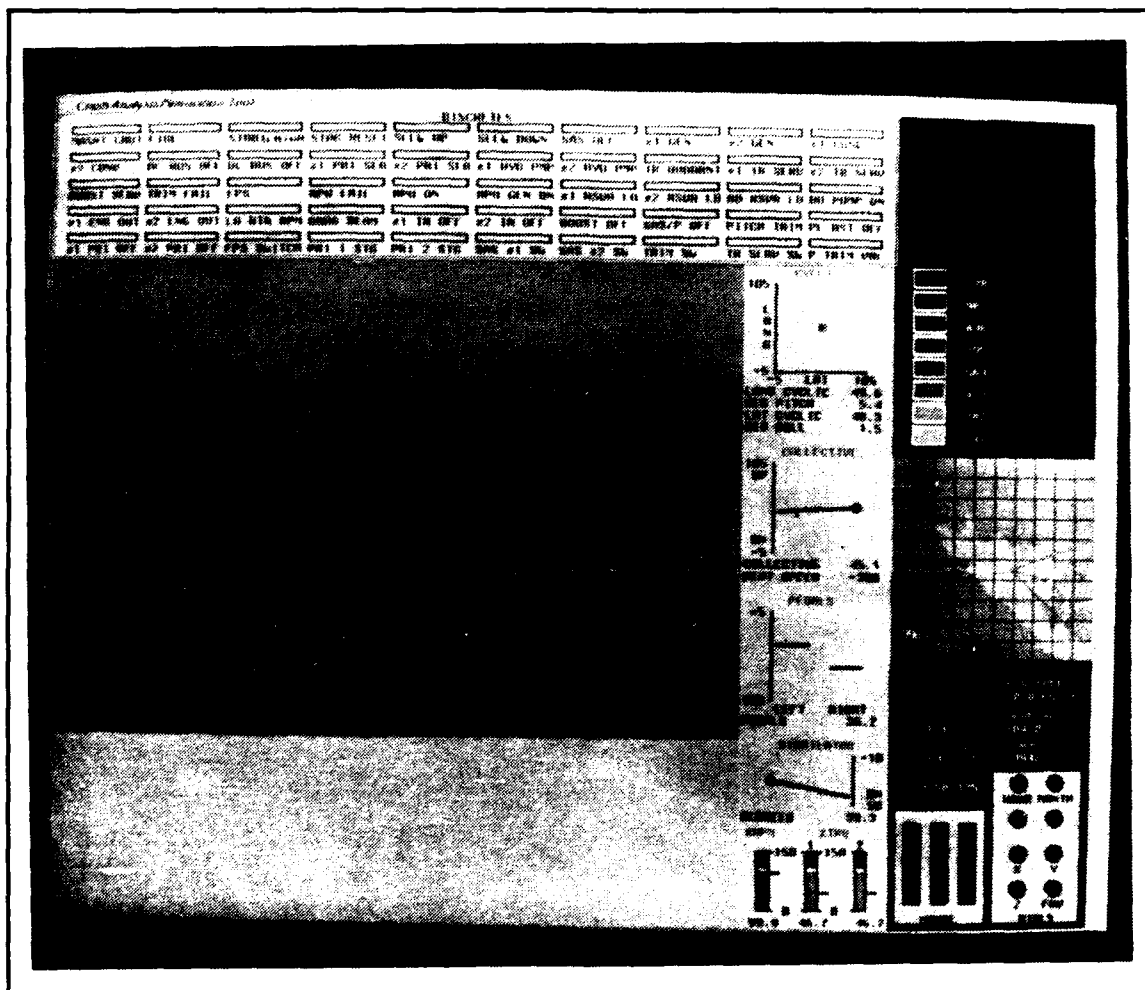


Figure 8 Display of the COBRA Helicopter Model

The MPS platform display routines use a standard C-code sequence to draw the polygons. Material color characteristics are first bound to the polygon with the **lmbind** command. Next, the polygon normal is specified for lighting calculations with the **n3f** command. The vertices of the polygon are then input to the graphics pipeline with the **v3f** command. A separate **v3f** command is issued for each vertex. The **v3f** commands are bracketed by a **bgnpolygon** and an **endpolygon** command. Figure 9 lists a representative block of code used to draw a jeep polygon in the MPS system.

```
/* left cab */  
lmbind(MATERIAL,JEEPOUT);  
n3f(pnjeep2);  
bgnpolygon();  
    v3f(pjeep2[0]);  
    v3f(pjeep2[1]);  
    v3f(pjeep2[2]);  
    v3f(pjeep2[3]);  
endpolygon();
```

Figure 9 MPS Polygon Drawing Code

A separate block of code similar to that in Figure 9 is used in the MPS platform drawing routines for each polygon drawn. This makes the drawing routines rather lengthy and is wasteful of memory space. As an example, the MPS routines for drawing a sixty eight polygon truck contain 750 lines of C-code and occupy 17745 bytes of memory. More efficient drawing routines were developed for the CAST system. The routines which draw the 185 polygon COBRA helicopter contain only 350 lines of code and occupy only 7236 bytes of memory.

2. CAST Helicopter Display Programs

A different array structure was implemented in the CAST software to store the helicopter polygons than was used in MPS to store platform polygons. The helicopter polygons are grouped by number of vertices and color. A three dimensional array structure of floating point numbers is utilized to store each group of polygons. The first array dimension indicates the number of polygons in the group. The second dimension of the array indicates the number of vertices in each polygon within the group. The last dimension is always three for the X, Y, and Z world coordinates of each vertex. As an example, the array `p3gcobra[16][3][3]` is an array which stores sixteen, three-vertex green polygons for the body of the COBRA helicopter. The normals for each group of polygons are stored in a two dimensional array structure. The normal array which corresponds to the `p3gcobra` array is `pn3gcobra[16][3]`.

A loop structure is used in CAST to draw each group of polygons. A `lmbind` command is issued prior to a loop to set the material color characteristics for the group of polygons. A single `n3f` and `bngpolygon-v3f-endpolygon` sequence is contained within a loop. Figure 10 shows a typical CAST drawing loop. This method of drawing polygons is much more efficient in terms of memory used.

Four separate routines are used in CAST to draw the COBRA helicopter. The main drawing routine is `draw_cobra`. This routine draws the body of the COBRA and calls `draw_tail_rotor`, `draw_main_rotor`, and `draw_tail_pipe` to draw the associated components. Accumulation rotation matrices are maintained for the main and tail rotors to provide turning rotor animation. The body of the COBRA is drawn with the center of gravity point (located in line with the main rotor shaft) at the origin. The nose of the helicopter faces along the positive X-axis. The main rotor,

tail rotor, and engine tail pipe are all defined about the origin and are then translated to the proper position on the body of the COBRA. The COBRA helicopter is defined in a world coordinate system which represents meters. The C programs which draw the COBRA helicopter are included in Appendix B.

```
/* draw the four-vertex, green polygons for the COBRA body */  
lmbind(MATERIAL,COBRABODY);  
for (i = 0; i < 90; i++)  
{  
    n3f(pn4gcobra[i]);  
    bgnpolygon();  
    v3f(p4gcobra[i][0]);  
    v3f(p4gcobra[i][1]);  
    v3f(p4gcobra[i][2]);  
    v3f(p4gcobra[i][3]);  
    endpolygon();  
}
```

Figure 10 CAST Polygon Drawing Code

In developing the CAST software, the COBRA helicopter was first added to the MPS system by following the procedure outlined in the Fichten and Jennings thesis [Ref. 7:p. 108-110]. At this point, the helicopter acted as a FOG-M. The MPS menu, user interface, and vehicle position update coding was then modified so that the flight of the helicopter could be displayed and analyzed.

E. USER POPUP MENU INTERFACE MODIFICATIONS

The basic menu structure of MPS was utilized while developing the CAST system. Several of the menus were removed because there are no ground vehicles in CAST. Similar menus were consolidated and three new menus were added to provide options not available in MPS.

1. Popup Menus Deleted

A total of ten popup menus were deleted from the MPS program during the development of CAST. When a platform is added for display in MPS, the user must provide an initial speed and direction for the platform. This is done with popup menus. The initial speed and direction for a helicopter added for display in CAST are determined from the helicopter's flight data file. This allowed the removal of the GROUND SPEED, FLYING SPEED, and DIRECTION popup menus.

Since ground vehicles were removed for the CAST implementation, the menus to control the operation of ground vehicles (OPERATE DRIVE) and to add a default convoy of vehicles (DEFAULT_MENU) were deleted. The speed of an aircraft displayed in CAST is determined from its flight data. The user can not control the aircraft speed. Therefore, the CHANGE_SPEED_MENU popup was removed.

MPS provides an ADVANCED menu which allows the user to select options which demonstrate various graphics capabilities of the IRIS workstation. This feature is not needed for a crash analysis simulator, so the ADVANCED menu was eliminated.

The MPS system allows the user to temporarily detach from operation while flying a FOG-M and store the state of the system in a data file. The user can then

later resume operation at the same state as when he detached from operation. To implement this option, MPS uses three separate menus (OPERATE_FLY_ONE, OPERATE_FLY_TWO, and OPERATE_FLY_THREE). The detach capability was not necessary for CAST and was removed. This allowed the deletion of the OPERATE_FLY_ONE and OPERATE_FLY_THREE popup menus. The OPERATE_FLY_TWO popup menu was retained and renamed OPERATE_FLY.

Two terrain selection menus are used in MPS. If the networking capability is not activated or is activated and only one process is active, an area of terrain or new terrain database can be selected. If networking is activated and more than one process is active, terrain and data base selection is not allowed. The SELECT_AREA_ONE popup menu allows terrain and database selection while the SELECT_AREA_TWO popup menu does not. Networking was removed for the CAST implementation so the SELECT_AREA_TWO popup menu was also removed. The SELECT_AREA_ONE menu was kept and renamed SELECT_AREA.

2. Popup Menu Options Deleted

The MPS popup menus provide several options which can be selected, but which do nothing. These options are indicated by being displayed in lower case letters in the menus. Options which have an effect on the system are displayed in upper case letters in the menus. Although an unuseable option serves as a reminder that the operation is not currently available, it was decided that such an option violates good user interface principles. The user should only be presented with options which affect the system. All menu options which have no effect on the system were therefore removed.

The options to add individual ground vehicles and missiles were removed from the popup menus as well as the options to add a default set of platforms and to save the platforms in a data file. The missile tracking option, resize option, and change speed options were deleted from the OPERATE_FLY popup menu.

3. Popup Menus Added

Three popup menus were added for the CAST implementation. These menus are selected as roll-off-the-side menus.

A frame delay menu (DELAY) was added to allow the user to control the rate at which the frames of flight data are animated. The user can select a one, two, or five second delay between frames to aid in analyzing the data displayed. The user can also select a single step option to manually step forward and backward one frame at a time through the flight data. This allows the user to study the graphical display of a flight data frame for an indefinite period of time. The user can return to the normal frame display rate (no delay between frames) at any time. The DELAY popup menu can be selected from the MAIN-ONE, MAIN_TWO, MAIN_THREE, MAIN_FOUR, or OPERATE_FLY popup menus.

A menu was added (EYE) to provide variable placement of the eyepoint for display of the aircraft over the three dimensional terrain. The user can select an eyepoint inside the cockpit of the aircraft or outside of the aircraft. Within the cockpit, the user can select either the pilot station or the copilot station for the eyepoint. This allows the user to watch the flight from the same perspectives as the pilots who were involved in a mishap. The user can also place the eyepoint outside of the aircraft and watch the aircraft as it flies over the terrain. This is especially useful when analyzing

midair collisions. Section F of this chapter provides details of the control of the eyepoint when it is placed outside of the aircraft. The EYE popup menu is selectable from the OPERATE_FLY menu.

An altitude option menu (ALTITUDE) was added which provides for three means of displaying the aircraft above the ground. If the terrain database is not available for the area in which the mishap occurred, any terrain database can be used with the constant altitude option selected. This displays the aircraft at a constant altitude of 200 meters above ground level. If the proper terrain database is available, the user can display the aircraft utilizing the actual altitude data recorded in the flight data file. This can result in an unstable display because of large variances in altitude values between frames. A smoothed altitude option is available to provide a more stable display of the aircraft. Section I of this chapter discusses altitude smoothing. The ALTITUDE menu can be selected from the OPERATE_FLY popup menu.

F. USER CONTROL INTERFACE MODIFICATIONS

The user control interface for both the MPS and CAST system is through the mouse and dial box. The control usage was extensively modified during the development of CAST.

1. Mouse Control Modifications

The mouse control functions for the MPS system are depicted in Figure 11. The controls have the following utilizations:

1. Vertical mouse movement - This controls the panning (side to side movement) of the camera in a FOG-M which is not actively tracking a ground target.

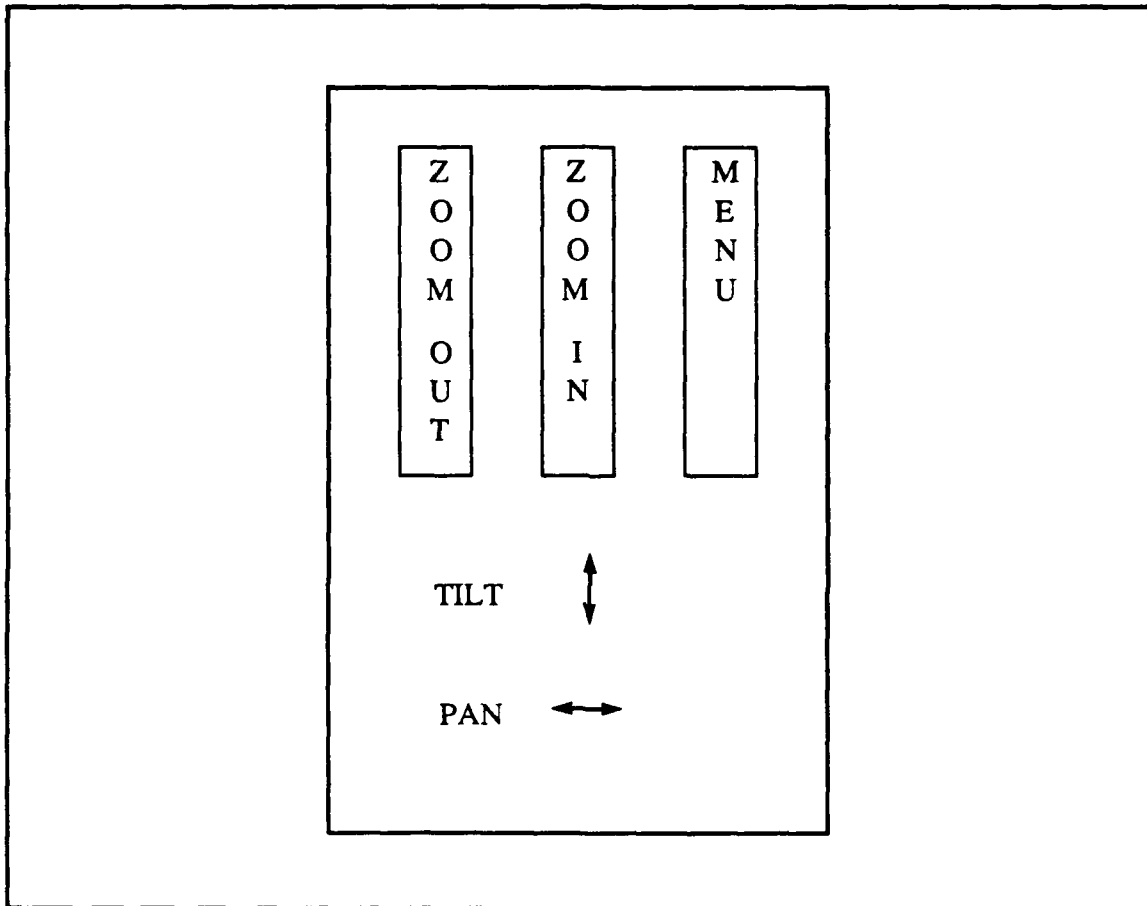


Figure 11 MPS Mouse Controls

2. Horizontal mouse movement - This controls the tilt angle (up and down movement) of the camera in a FOG-M which is not actively tracking a ground target.
3. Left mouse button - This is used to increase the field of view as seen from a driven platform or the FOG-M camera.
4. Middle mouse button - This control decreases the field of view as seen from a driven platform or the FOG-M camera.
5. Right mouse button - This button is used to display popup menus on the screen.

Since the FOG-M was not to be used in CAST, the horizontal and vertical mouse movement control interfaces were disabled. A variable field of view capability was to be included in the CAST system. It was decided to control the field of view with a dial rather than with the mouse. This freed the left and middle mouse buttons for other uses. The control interfaces provided by the mouse in the CAST system vary depending upon whether the user is operating under the manual frame step mode or not. The control functions of the mouse in CAST are depicted in Figure 12. These controls are utilized as follows:

1. Left mouse button - When operating under the manual frame step mode, this button displays the previous frame of flight data.
2. Middle mouse button - When operating under the manual frame step mode, this button displays the next frame of flight data.
3. Right mouse button - This button is used to display popup menus on the screen.

2. Dial Box Control Modifications

The dial box control functions in MPS vary depending upon the type of vehicle being operated by the user. The dial functions provided when a ground vehicle is being operated are shown in Figure 13. Figure 14 shows the dial functions provided when the operator is controlling a missile.

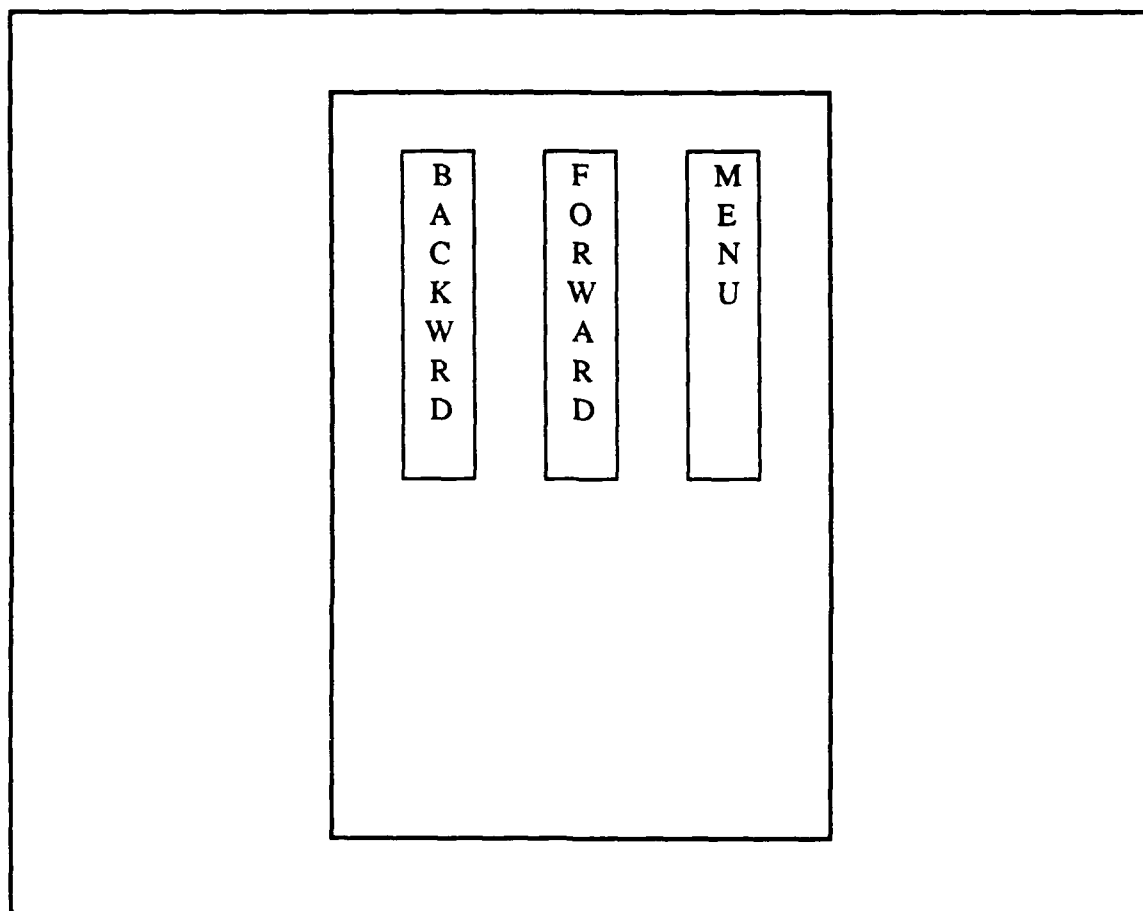


Figure 12 CAST Mouse Controls

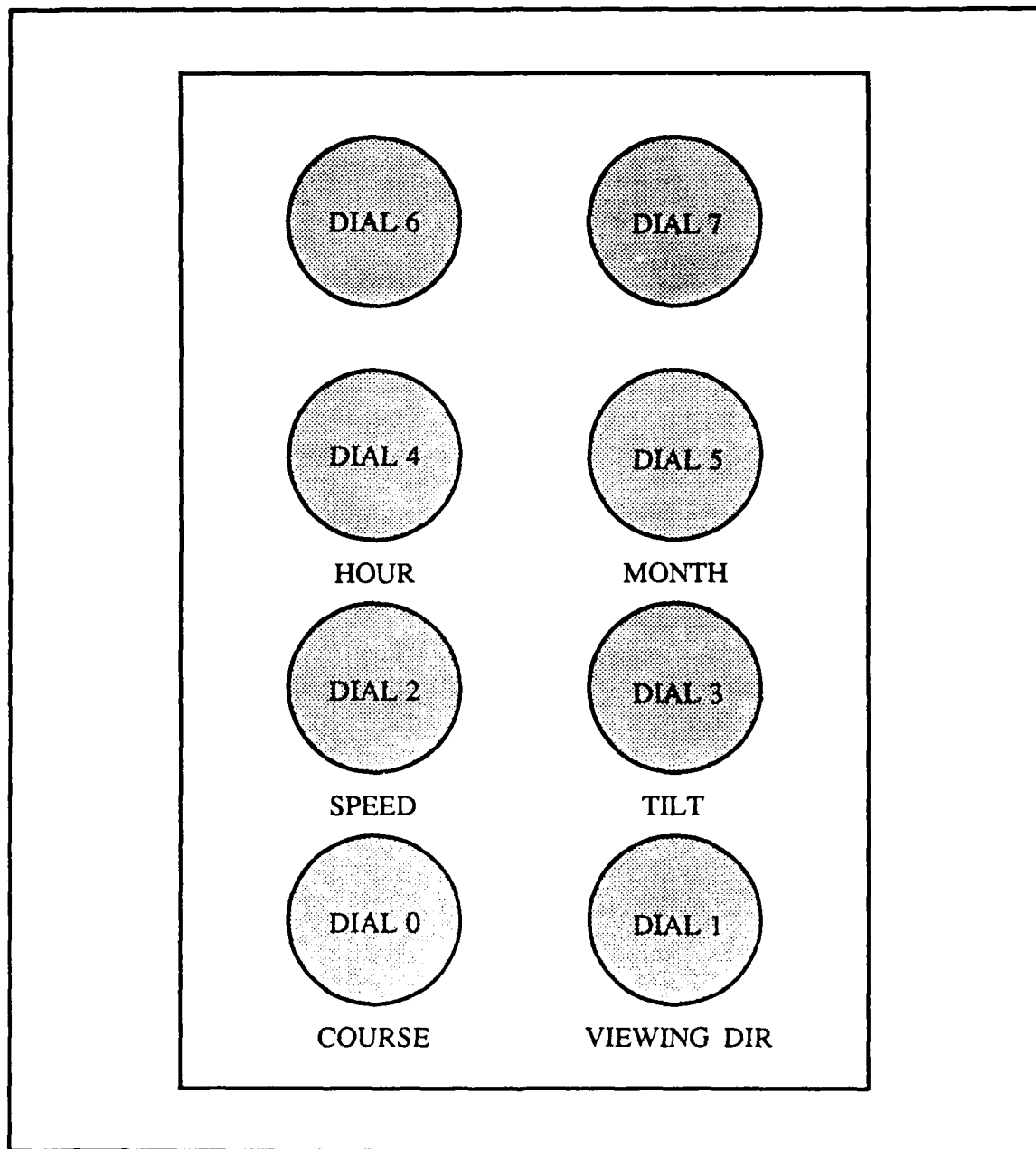


Figure 13 MPS Dial Box Controls for Driving a Vehicle

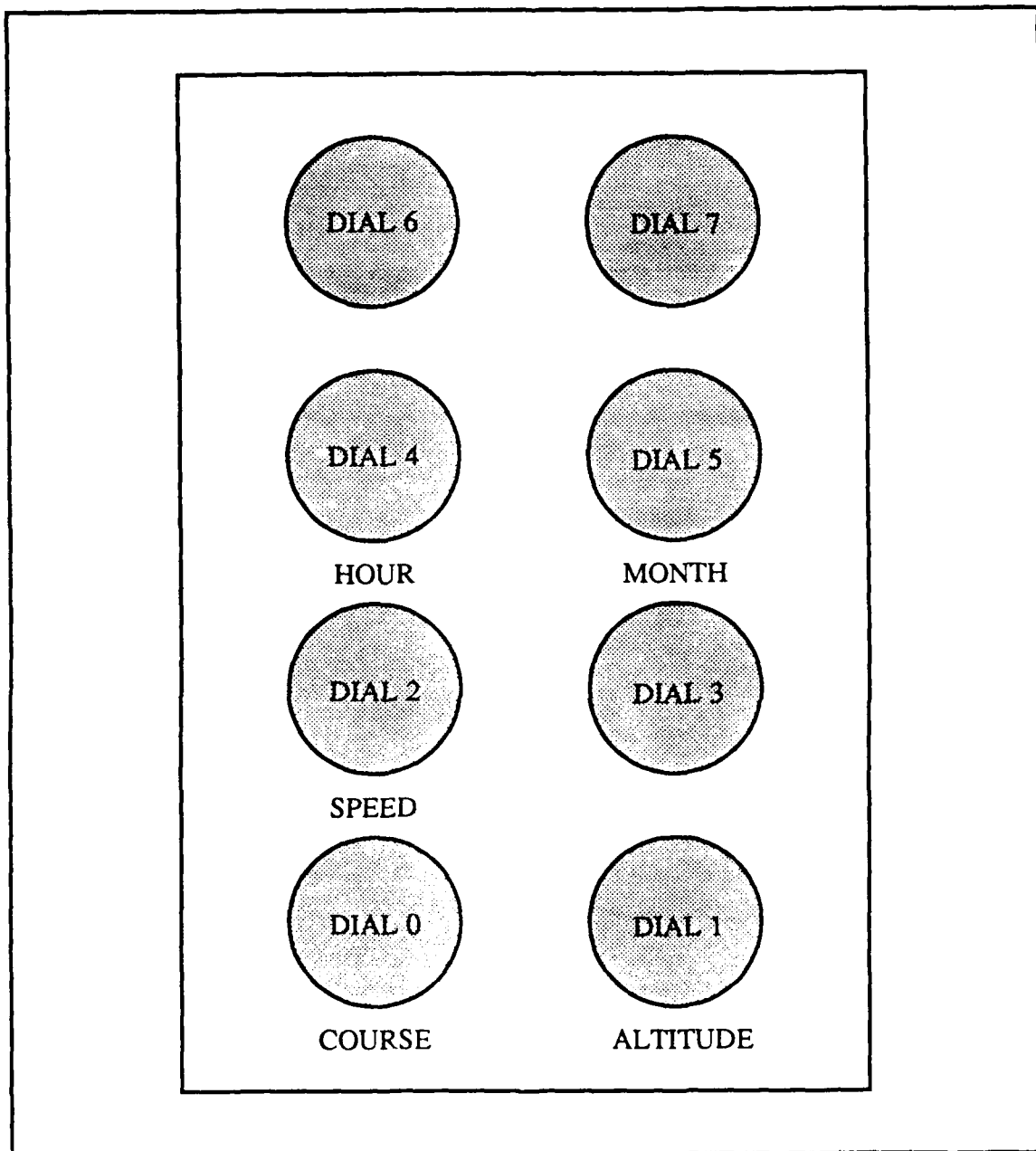


Figure 14 MPS Dial Box Controls for Flying a Missile

In the MPS system, the dials are utilized for:

1. DIAL 0 - When operating a ground platform, this dial controls the vehicle's course. When operating a missile and not tracking a ground platform, the dial controls the missile's course. When operating a missile that is tracking a ground platform, the dial is not used.
2. DIAL 1 - When operating a ground platform, this dial controls the viewing direction of the vehicle driver. When operating a missile and not tracking, the dial controls the missile's altitude. When operating a missile that is tracking a ground platform, the dial is not used.
3. DIAL 2 - This dial is used to control the speed of all platforms.
4. DIAL 3 - When operating a ground platform, this dial controls the viewing elevation of the vehicle driver. The dial is not used when operating a missile.
5. DIAL 4 - This dial is used to control the time of day for lighting of the three dimensional scene for all platforms.
6. DIAL 5 - This dial is used to control the month of the year for lighting of the three dimensional scene for all platforms.
7. DIAL 6 - This dial is not used.
8. DIAL 7 - This dial is not used.

Since the aircraft displayed in the CAST system were to be flown from flight data files, speed, course, and altitude controls were not needed. Controls were needed to vary the viewing direction, field of view, and the month and time of day for lighting purposes. The dial controls available to the user as implemented in CAST vary depending upon whether the eyepoint is placed inside an aircraft or outside of an aircraft. Figure 15 shows the dial box controls when the eyepoint is inside an aircraft. Figure 16 shows the controls when the eyepoint is moved outside of an aircraft.

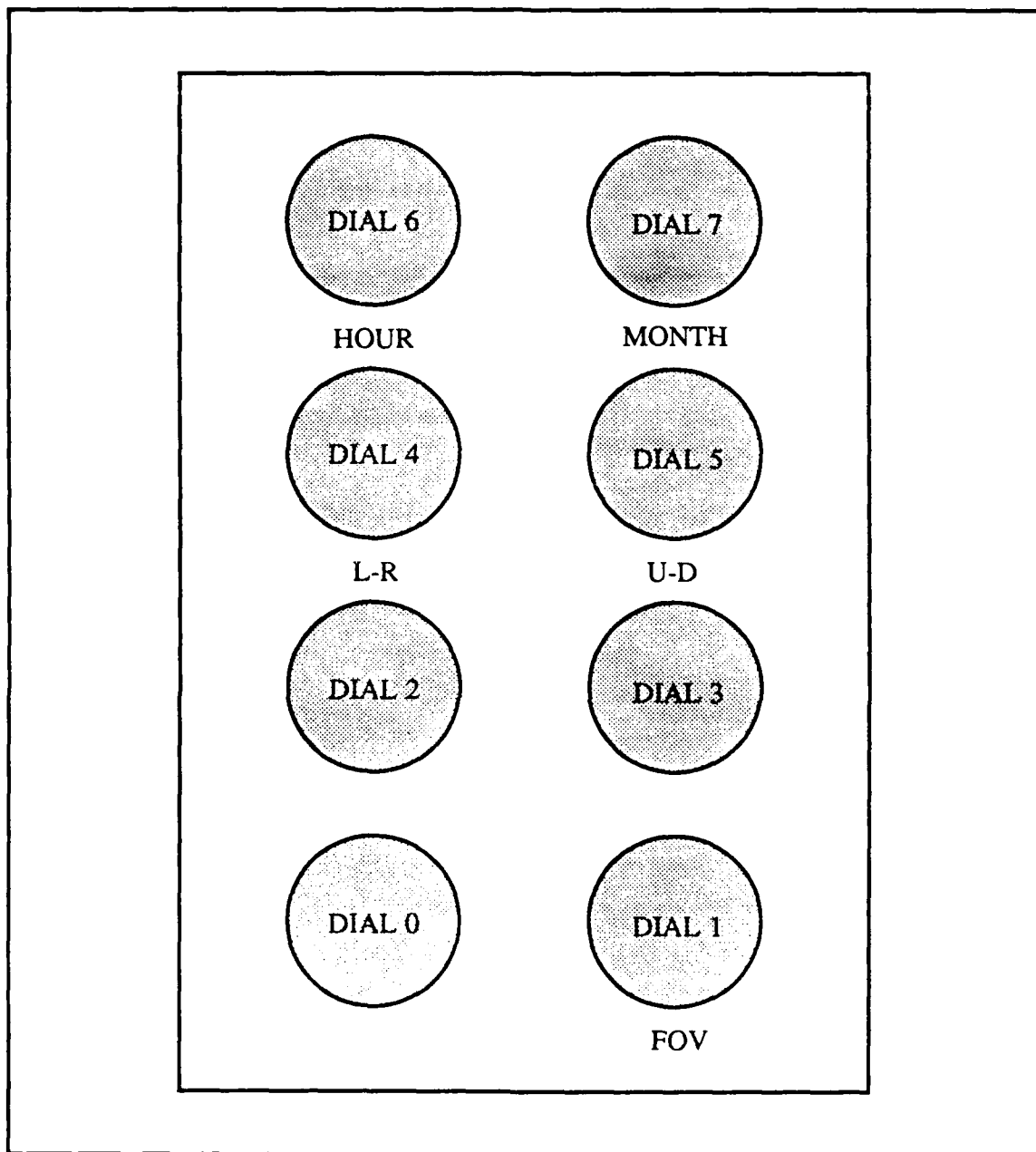


Figure 15 CAST Dial Box Controls With Eyepoint Inside Aircraft

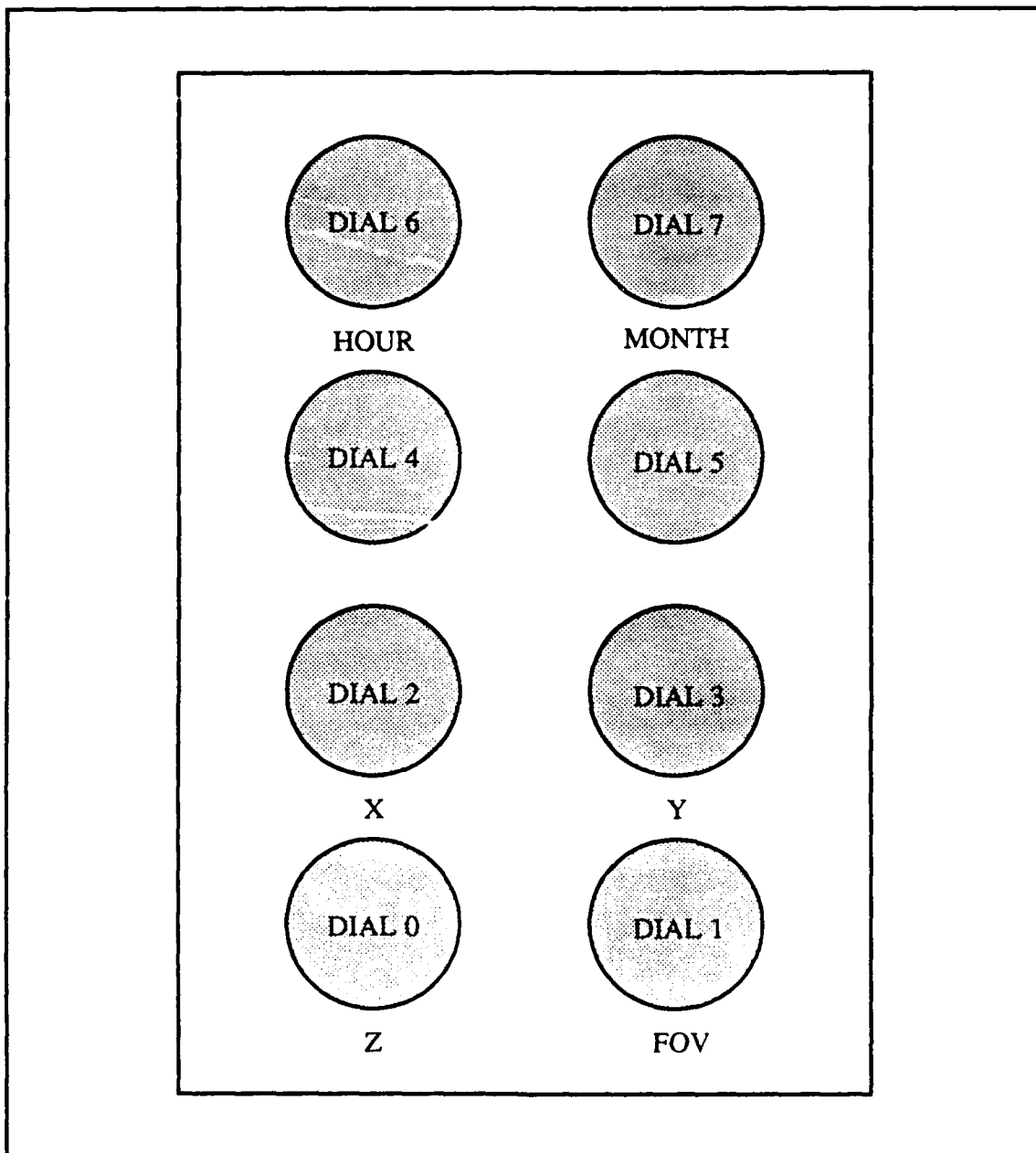


Figure 16 CAST Dial Box Controls With Eyepoint Outside Aircraft

The dial box controls provide the following functions in CAST:

1. DIAL 0 - When the eyepoint is outside an aircraft, this dial moves the eyepoint along the Z-axis of the aircraft world coordinate system. The eyepoint can be placed anywhere within positive and negative fifty meters of the center of the aircraft along the Z-axis. When the eyepoint is inside an aircraft, this dial is not used.
2. DIAL 1 - Regardless of where the eyepoint is located, this dial controls the field of view. The field of view can be varied from ten degrees to one hundred degrees.
3. DIAL 2 - When the eyepoint is outside an aircraft, this dial moves the eyepoint along the X-axis of the aircraft world coordinate system. The eyepoint can be placed anywhere within positive and negative fifty meters from the center of the aircraft along the X-axis. When the eyepoint is inside an aircraft, this dial is not used.
4. DIAL 3 - When the eyepoint is outside an aircraft, this dial moves the eyepoint along the Y-axis of the aircraft world coordinate system. The eyepoint can be placed anywhere within positive and negative fifty meters from the center of the aircraft along the Y-axis. When the eyepoint is inside an aircraft, this dial is not used.
5. DIAL 4 - When the eyepoint is inside an aircraft, this dial changes the lateral viewing position as seen by a pilot. The range for the lateral viewing position is plus and minus one hundred twenty degrees relative to the nose of the aircraft. When the eyepoint is outside of an aircraft, this dial is not used.
6. DIAL 5 - When the eyepoint is inside an aircraft, this dial changes a pilot's viewing elevation. The range for the viewing elevation is plus and minus ninety degrees relative to the centerline of the aircraft. When the eyepoint is outside of an aircraft, this dial is not used.
7. DIAL 6 - This dial is used in all cases to control the time of day for lighting of the three dimensional display.
8. DIAL 7 - This dial is used in all cases to control the month of the year for lighting of the three dimensional display.

Figure 17 shows the graphics displayed with the eyepoint inside the COBRA helicopter. The eyepoint is placed at the pilot's station with the pilot looking to his left. Figure 18 shows the graphics displayed with the eyepoint outside an aircraft. Here, a flight of three COBRA helicopters is being viewed.

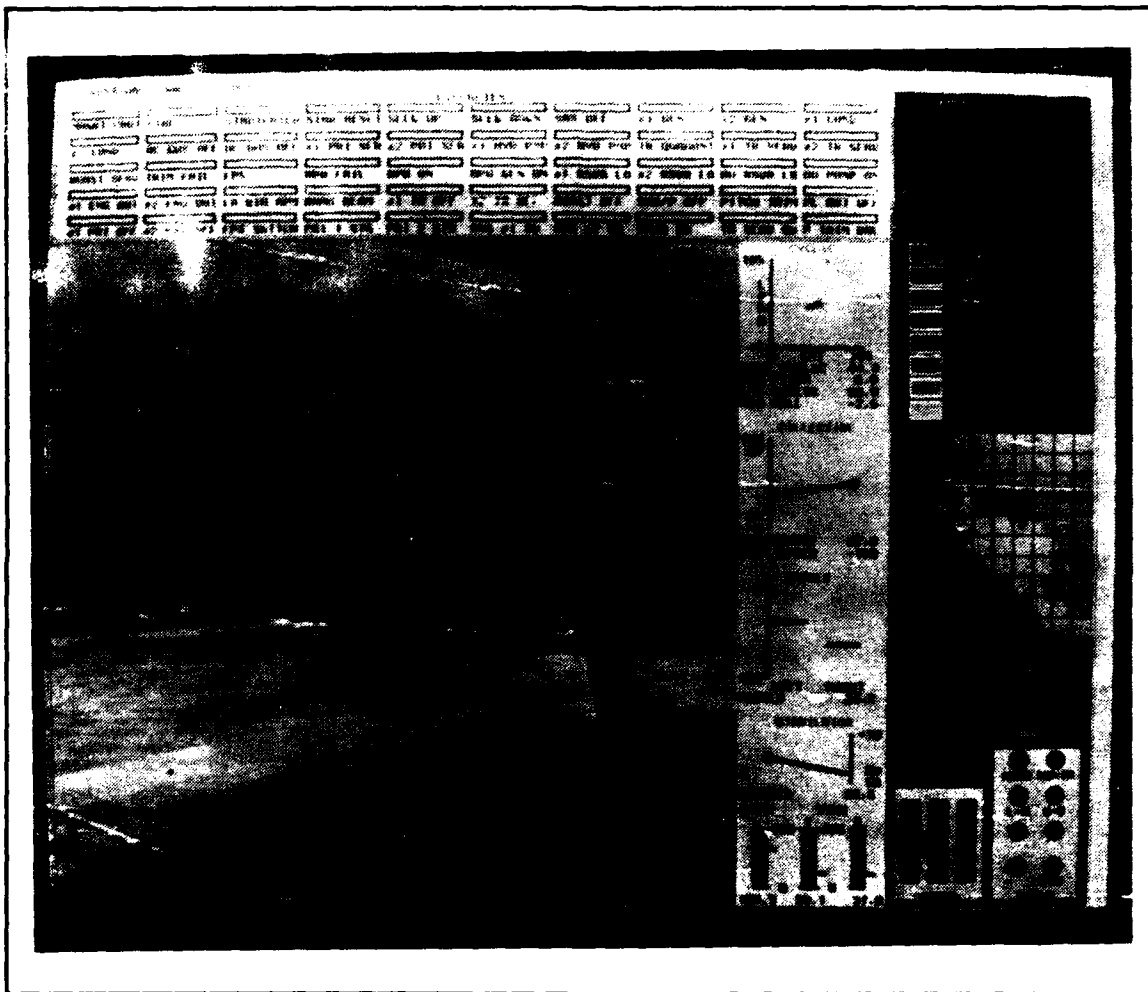


Figure 17 View From Inside a COBRA Helicopter

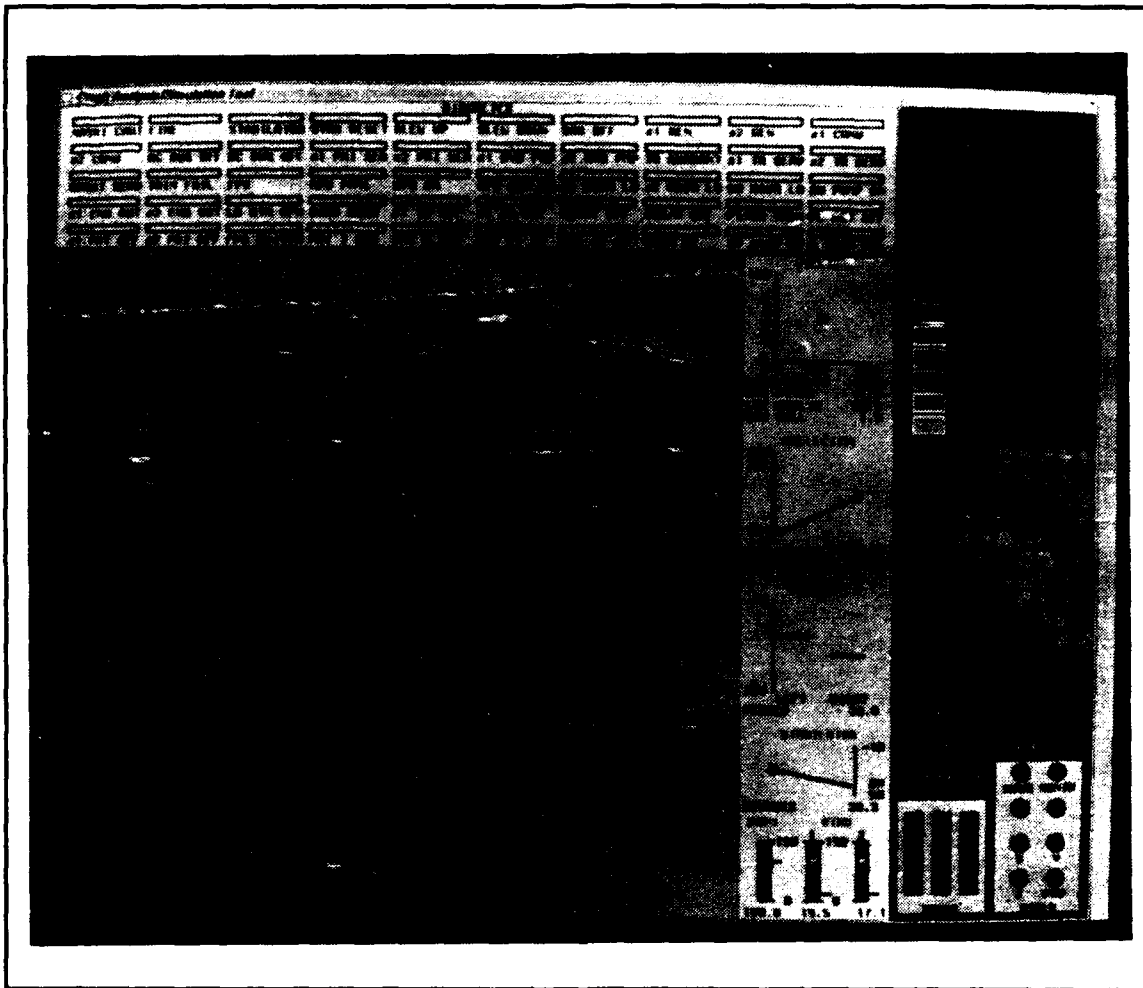


Figure 18 View of Three COBRA Helicopters in Formation Flight

3. Graphics Displayed In The INDWIN Graphics Window

A graphical representation of the mouse and dial box are displayed in the INDWIN graphics window. The mouse buttons and dials are labeled to indicate which controls are currently active. Active controls have a label, inactive controls do not.

In addition, selected flight data is displayed in this window. The combined time and line (frame) counter parameters are displayed as are the yaw attitude (heading), airspeed, and altitude mean sea level (MSL) parameters. The altitude above ground level (AGL) as computed from the terrain data is displayed. The current field of view is also shown.

G. PLATFORM POSITION UPDATE MODIFICATIONS

1. Position Update In MPS

The MPS system uses the algorithms developed in VEH to update platform positions for both the two and three dimensional displays [Ref. 7:p. 21]. The displacement of a platform is calculated as a function of the speed of the platform and the elapsed time since the last update of the platform's position. The direction of the displacement is based upon the platform's compass heading. The pitch and roll of the ground platforms are calculated in relation to the slope of the terrain over which the platforms are traversing. Flight dynamics are not incorporated into the display of the FOG-M. All changes in a missile's altitude and heading are done in a level flight attitude which eliminates the need for pitch and roll calculations.

As the user controls a platform in MPS, any changes in the platform's speed and heading are recorded in the **Vehicle** data record for the platform. The linked list of **Vehicle** data records is scanned during the platform position update cycle. The position of each platform is updated based upon the data currently stored in the data records.

2. Position Update In CAST

The update of an aircraft's position in the CAST system is based upon the aircraft's flight data file. During the position update cycle, the linked list of **Vehicle** records is scanned. Each **Vehicle** record contains a pointer into the flight data record linked list for the aircraft. This points to the current flight data record to be used for display of the aircraft. The flight parameters stored in the current flight data record are used to update the **Vehicle** record. Since the parameters are recorded at varying rates by the flight recorder, some fields in the **Vehicle** record are not updated on every cycle. These fields retain the values from previous updates.

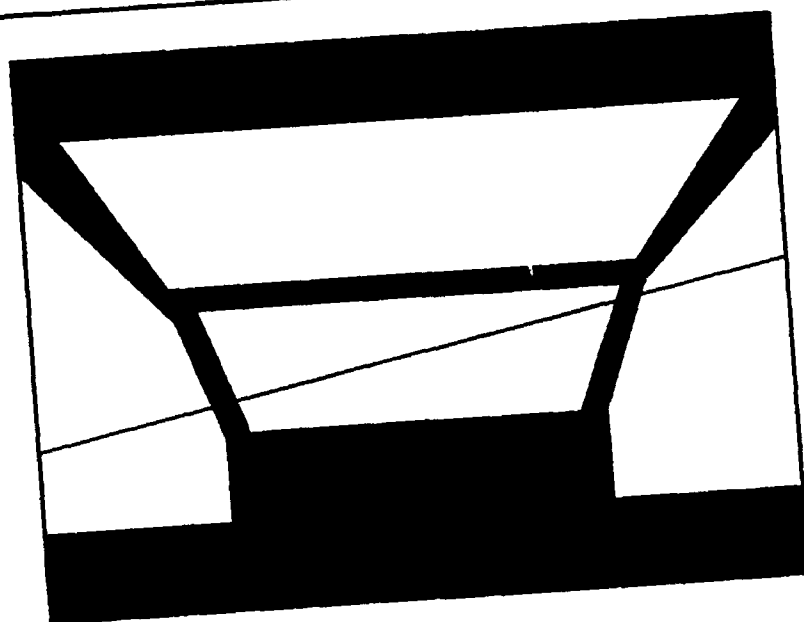
The displacement calculation for position update is done differently in CAST than in MPS. Each frame of flight data in the CAST system represents a one eighth second time interval. Elapsed time calculations between display of graphics frames are no longer required. The distance traveled by an aircraft between display of graphics frames is simply calculated by multiplying the aircraft's speed by one eighth of a second.

In CAST, aircraft flight dynamics are modeled from the roll, pitch, and yaw attitude flight parameters recorded in the flight data. The roll, pitch, and yaw of an aircraft are done around the center of gravity point of the airframe.

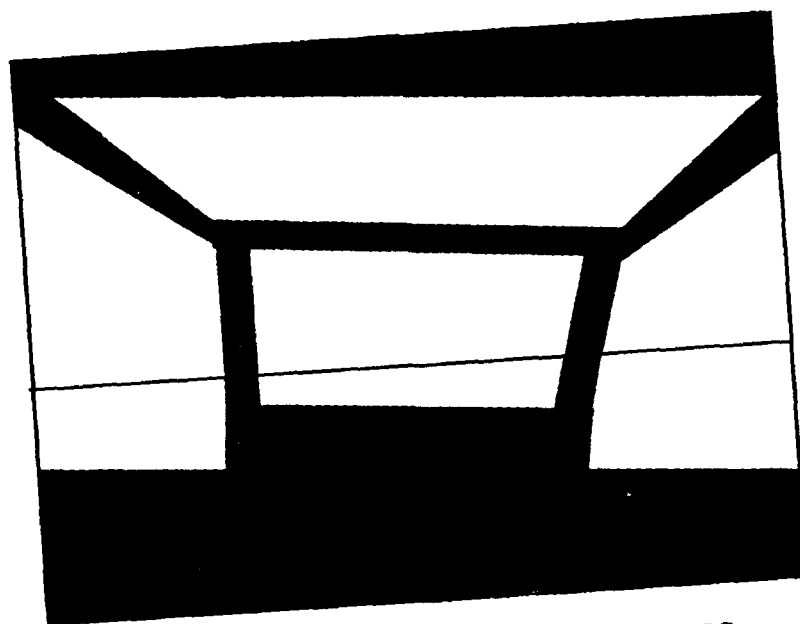
H. MODIFICATION OF VIEWING PERSPECTIVE

The viewpoint in MPS is fixed with respect to the platform body coordinate system. The IRIS graphics software requires that the viewpoint be expressed in terms of the graphics system's coordinate system. By placing the viewpoint at the origins of the X and Z-axes of the platform body coordinate system, a relatively satisfactory viewing perspective was achieved. This approach to the placement of the viewpoint presented two problems during the development of CAST. When the viewpoint was translated away from the X and Z-axis origins for placement at the cockpit flight stations; and the aircraft was rotated, the viewpoint did not rotate with the aircraft. Instead, the aircraft rotated around the viewpoint. Additionally, the horizon was not properly displayed as the aircraft pitched and rolled. Figure 19(a) shows a proper representation of how the horizon should be displayed while in a right banking turn. Notice that the cockpit of the aircraft remains level while the horizon is tilted. Figure 19(b) shows how the right banking turn was displayed using the MPS viewpoint scheme. Here, the horizon remains level and the cockpit is tilted.

To correct these problems, a coordinate system transformation was required between the viewpoint expressed in the body coordinate system of the aircraft and the coordinate system of the IRIS workstation. The required transformation was developed by another thesis student who was also using the MPS system as a baseline for his thesis work [Ref. 12:p. 18-20]. The program which he developed to perform the coordinate system transformation was modified to run in the CAST system. This code can be found in Appendix B. The result was a viewing perspective which accurately models what is seen from an aircraft.



(a) Proper Display of the Horizon in CAST



(b) Improper Display of the Horizon in MPS

Figure 19 Display of the Horizon in CAST and MPS

I. ALTITUDE SMOOTHING

When the actual altitudes recorded in a flight data file are used for the display of an aircraft, the display becomes somewhat unstable. The aircraft appears to bounce up and down on screen. This is due in part to the inherent inaccuracy of the barometric altimeter from which the flight data is recorded. The relatively slow sample rate for recording altitude data also is a factor. Since the altitude parameter is only recorded twice per second, the CAST system uses each altitude for the display of four consecutive frames when the actual altitude option is selected from the ALTITUDE popup menu. On the display of every fourth frame, a new altitude is used. Since the altitude samples are taken at half second intervals, a significant change in altitude can occur between the samples. To overcome the unstable display, a smoothed altitude option was implemented in CAST.

1. Altitude Smoothing Algorithms

Several algorithms were tested for smoothing the altitude data. All of the algorithms involved averaging several consecutive altitude samples. The number of samples averaged and the weight assigned to the altitude for the current display frame were varied in the algorithms. The goal for the smoothing algorithm was to minimize the difference between consecutive smoothed altitude values while keeping the smoothed altitudes close to the actual recorded values.

Six smoothing algorithms were tested. Three of the algorithms averaged three consecutive altitudes and three averaged five consecutive altitudes. In each set of algorithms, the weight assigned to the altitude for the current frame being displayed was set at one, two, and three. The algorithms used are shown in Table 6.

TABLE 6 ALTITUDE SMOOTHING ALGORITHMS

NAME	ALGORITHM
s-3-1	$(ALT_{cf-1} + (1 * ALT_{cf}) + ALT_{cf+1}) / 3$
s-3-2	$(ALT_{cf-1} + (2 * ALT_{cf}) + ALT_{cf+1}) / 4$
s-3-3	$(ALT_{cf-1} + (3 * ALT_{cf}) + ALT_{cf+1}) / 5$
s-5-1	$(ALT_{cf-2} + ALT_{cf-1} + (1 * ALT_{cf}) + ALT_{cf+1} + ALT_{cf+2}) / 5$
s-5-2	$(ALT_{cf-2} + ALT_{cf-1} + (2 * ALT_{cf}) + ALT_{cf+1} + ALT_{cf+2}) / 6$
s-5-3	$(ALT_{cf-2} + ALT_{cf-1} + (3 * ALT_{cf}) + ALT_{cf+1} + ALT_{cf+2}) / 7$

cf = current frame

2. Algorithm Testing

A random sample of 200 consecutive altitudes was extracted from a flight data file to test the algorithms. Within the sample block of altitudes, the maximum change between altitudes was 145.0 feet. The average change between altitudes was 35.8 feet. Table 7 shows the results of the algorithm testing. Listed in the table are the maximum and average changes between smoothed altitude values and the maximum and average differences between the smoothed altitude values and the actual altitude values. After comparing the results from the six algorithms, the s-5-2 algorithm was selected for the implementation of the altitude smoothing option. This algorithm provides the best balance between minimizing the change between smoothed altitude values and keeping the smoothed values close to the recorded altitudes.

TABLE 7 SMOOTHING ALGORITHM RESULTS

NAME	ALGORITHM					
	s-3-1	s-3-2	s-3-3	s-5-1	s-5-2	s-5-3
Maximum change between smoothed altitudes	49.0'	59.0'	70.0'	41.0'	39.0'	51.0'
Average change between smoothed altitudes	14.1'	14.8'	17.8'	11.3'	12.5'	14.9'
Maximum difference between smoothed & recorded altitudes	87.0'	65.0'	52.0'	92.0'	77.0'	66.0'
Average difference between smoothed & recorded altitudes	21.1'	15.9'	12.6'	22.8'	19.0'	16.3'

3. Additional Altitude Smoothing

The use of the s-5-2 smoothing algorithm only partially handles the altitude smoothing task. The algorithm is applied only to every fourth frame of flight data. This still leaves three frames to be smoothed between each pair of s-5-2 smoothed altitudes. The altitudes for these frames are calculated by successively adding one fourth of the difference between the two surrounding s-5-2 smoothed altitudes to the first altitude in the s-5-2 smoothed pair. The graph in Figure 20 shows the results of smoothing a four second block of altitude data. The altitude smoothing is done in CAST when an aircraft is added to the system. The smoothed altitude values are stored in the **Flt_data_rec** records. When the smoothed altitude option is selected, these values are used to update the **Vehicle** records instead of the altitudes recorded by the flight data recorder. The altitude smoothing code can be found in Appendix B.

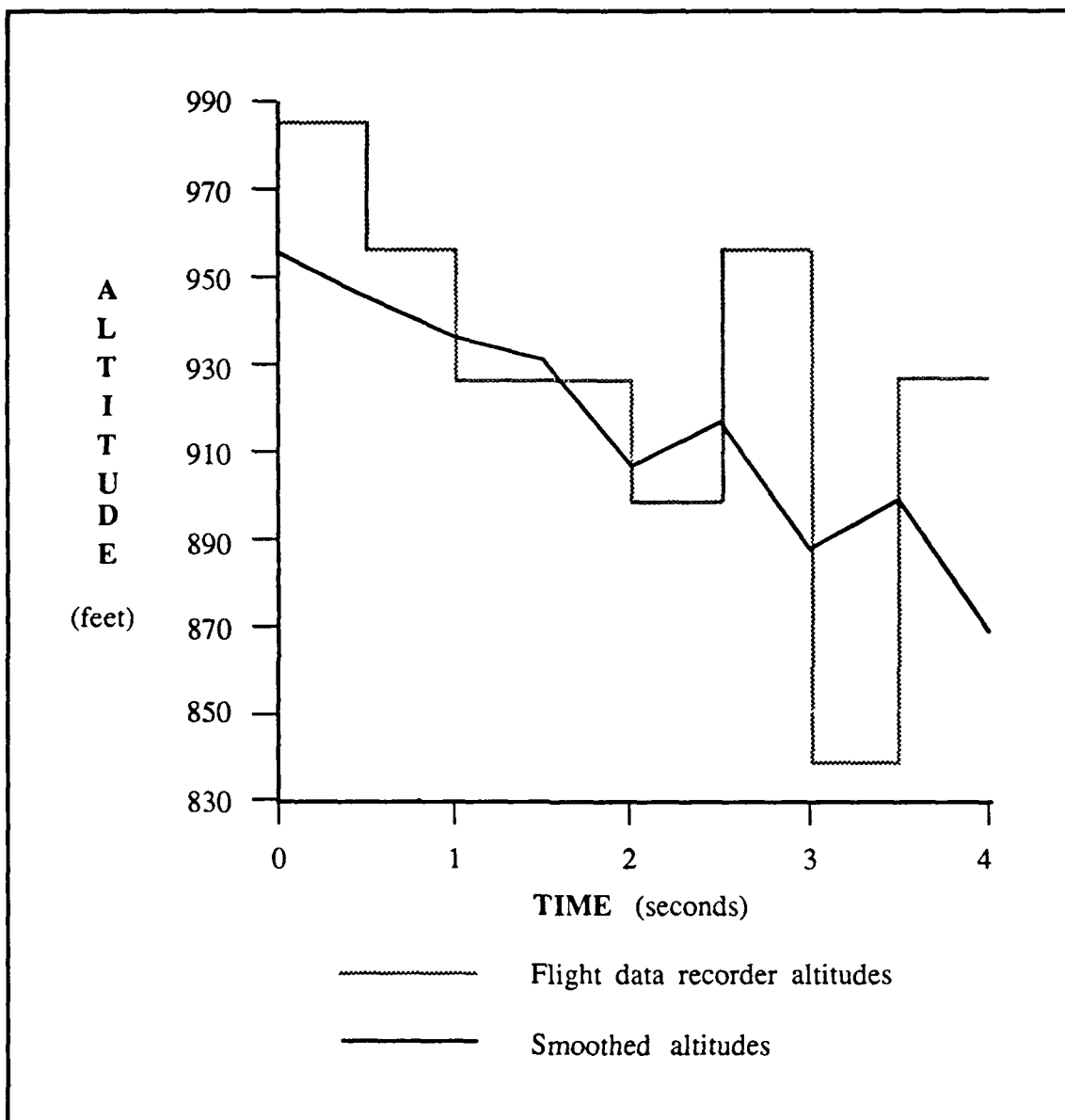


Figure 20 Altitude Smoothing

V. CONCLUSIONS AND RECOMMENDATIONS

A. SYSTEM PERFORMANCE

One goal of all research conducted in the Graphics and Video Laboratory at the Naval Postgraduate School is to develop graphics systems which run in real time. Real time implies that the frame display rate is fast enough to provide smooth, continuous animation. The standard against which real time performance is based is the projection of motion picture film. Motion picture film is projected at a rate of twenty four frames per second. Because of the complexity of the systems developed in the Graphics and Video Laboratory, the standard is rarely met. Frame update rates are achieved, however, which do provide a reasonably smooth animation of objects displayed.

The frame update rate for a system varies depending upon the complexity of the graphics being displayed. The frame update rate for the MPS system can vary anywhere between less than one to fifteen frames per second depending upon the number of vehicles displayed, the terrain detail, and the field of view [Ref. 7:p. 63].

The performance goal for the CAST system was to accomplish a frame update rate of eight frames per second. Since the flight data is recorded at a rate of eight samples per second, a frame update rate of eight frames per second would provide a true real time visualization of the flight recorder data. An average frame update rate of six frames per second was achieved in the CAST system. This equates to a seventy five percent real time display of the data. IRIS system statistics were monitored during the testing of CAST. Graphics drawing routines accounted for over ninety

percent of the system usage. It is doubtful that the CAST program can be optimized for a faster display rate, but a seventy five percent real time display rate is very good.

B. CAST SYSTEM LIMITATIONS AND FUTURE RESEARCH

The overall goal of this research was to develop a prototype crash analysis simulator. Since the simulator developed is a prototype, it has several limitations. Future work can be performed to solve these limitations and provide an even more valuable tool for the investigation of aircraft mishaps.

1. Aircraft Placement And UTM Grid Coordinates

When an aircraft is added to the system in the current implementation of CAST, the operator selects a position on the two dimensional terrain map for the initial placement of the aircraft. This does not always allow the helicopter to follow the path over the terrain in which the mishap occurred. The present flight data recorders do not record position location data, however, the UTM grid coordinates of the crash site are known. A routine can be added to CAST which computes the flight path of an aircraft based upon the known coordinates of the crash site. The flight data records can be read in reverse order to accomplish this task. The airspeed and the back course heading in each record could be utilized to compute the UTM grid coordinates of the aircraft for the frame. The initial grid coordinates for the flight can then be found and the aircraft can be placed at those coordinates by the system.

During the development of CAST, concurrent work was being conducted by other students who were also using the MPS system as a basis for their research. One result of this research was the use of UTM grid coordinates for the positioning of

platforms [Ref. 13]. The procedures developed in this research can be incorporated into CAST to provide the UTM grid coordinate positioning of the aircraft.

2. Operational Area Boundaries

A 10km X 10km grid square is selected in both CAST and MPS for the operation of the platforms. When a platform reaches a boundary of this 10km X 10km area, it is considered to have crashed. This restricts the flight of an aircraft to the 10km X 10km area in CAST. The crashing of platforms at grid square boundaries has now been eliminated from MPS [Ref. 13]. Instead of crashing a vehicle at a grid square boundary, the system allows the operator to select an adjacent grid square for continued operation. This procedure can be placed into the CAST system.

3. Modeling Of Additional Aircraft

Only the AH-1 COBRA helicopter is currently modeled in the CAST system. Other aircraft need to be modeled. The first candidate aircraft for addition to CAST is the UH-60 BLACK HAWK helicopter since this is the only aircraft which presently is equipped with a flight data recorder. Additional aircraft which can be modeled include the AH-64 APACHE, CH-47D CHINOOK, and OH-58D KIOWA helicopters and the OV-1 MOHAWK airplane. All of these aircraft are scheduled for eventual installation of flight data recorders.

During the development of CAST, concurrent work was being conducted in the Graphics and Video Laboratory to develop a standard text-based file format for the display of three dimensional objects [Ref. 14]. This file format should be incorporated into the CAST system for the display of the aircraft.

4. X-Y Plotting Of Flight Data

The PC based crash analysis system has the capability of plotting any eight of the sixty four flight data parameters against time on an X-Y plot. This feature was not implemented in the prototype CAST system but should be added. A window can be defined in CAST for plotting flight data parameters. This window can be positioned in the same location as the MAPWIN graphics window. The **winpush** and **winpop** commands can be used to selectively display X-Y plotting of flight data or the three dimensional display of the flight.

5. Selective Frame Display

The current implementation of CAST allows continuous, automatic, forward linear progression through a flight data file or manual single stepping forward and backward through a file. An option can be added to CAST which will allow the user to select any frame of data for display.

6. Adverse Flight Condition Warning

The CAST system can be modified to provide a warning indication to the user any time that an adverse flight condition is encountered in the flight data. This indication can be in the form of a message and the ringing of the system bell. The current frame can be frozen while the user analyzes the flight data causing the warning.

C. CONCLUSIONS

The prototype crash analysis simulator developed for this study has great potential to become a valuable crash investigation tool. Future work in the

development of the CAST system will provide a tool that can be instrumental in the prevention of U.S. Army aviation accidents through the detailed analysis of mishaps with CAST.

APPENDIX A

CAST USER'S GUIDE

The purpose of this user's guide is to provide instructions for the operation of the CAST system which vary from the operation of the MPS system. A detailed guide on the operation of the MPS system can be found in Appendix A of Reference 7. The instructions provided here are for a routine analysis session with CAST.

A. PREPROCESSING FLIGHT DATA FILES

Before the CAST system can be used to analyze flight data, the Lotus 1-2-3 format flight data files of the aircraft involved in a mishap must be converted to CAST format files. This is done with the **convert** utility program. The filenames in the **fopen** commands of the **convert** program must be edited before converting each Lotus 1-2-3 file. The open for reading command must specify the name of the Lotus 1-2-3 flight data file to be converted. The filename specified in the open for writing command can be any filename not currently being used. Be sure to remember the CAST format filenames as they will be used when an aircraft is added to the CAST system. When a Lotus 1-2-3 format data file has been converted, the CAST format data file must be edited to insert the three digit tail number of the aircraft as the first item in the data file. The tail number must be on a separate line immediately before the flight data. The flight data files for the aircraft to be displayed by the system must reside in the same directory as the CAST executable code when the CAST system is started.

B. STARTING THE CAST SYSTEM

The cast system has three modes of operation:

1. Silent mode - When the silent mode of operation is selected, the system bell is not rung to indicate acceptance of user input. CAST is started in this mode with the command **cast -s**.
2. Test mode - When the test mode is selected, the opening rotating billboard is not displayed. CAST is started in this mode with the command **cast -t**.
3. Normal mode - This is the normal mode of operation without the silent or test modes selected. CAST is started in this mode with the command **cast**.

The silent mode and the test mode can be selected at the same time by listing both options in the opening command (ie. **cast -s -t**).

Once the opening command has been given and the 4Sight window positioning square has been displayed on the screen, open the CAST window to the maximum size possible.

C. SELECTING AN AREA OF OPERATION

When the CAST welcome screen is displayed, select the GO TO SELECT AN AREA option from the OPENING MENU popup. This will read in the default terrain database, display the two dimensional contour map of the terrain, and bring up the AREA SELECT MENU popup. If the proper terrain database is not currently being used, the TERRAIN DATABASE SELECTION roll-off-the-side menu can be selected to import the correct terrain database. Once the proper terrain contour map is displayed, a 10km X 10km area of the map can be selected. Use the SELECT AN AREA OF THIS MAP option of the AREA SELECT MENU popup to do this. When the area of operation has been selected, the aircraft to be displayed can then be

added. Use the GO TO MAIN MENU option of the AREA SELECT MENU popup to enter the aircraft.

D. ENTERING AIRCRAFT FOR DISPLAY

When the GO TO MAIN MENU option is selected, a two dimensional contour map of the area of operation is displayed along with the MAIN MENU popup. Aircraft can be added, deleted, and selected for operation with the MAIN MENU popup. To add an aircraft, select the ADD A PLATFORM option from the MAIN MENU popup. Then select the appropriate aircraft to add from the ADD A PLATFORM MENU popup. Currently only a COBRA helicopter can be selected from the ADD A PLATFORM MENU popup.

When a helicopter is selected from the ADD A PLATFORM MENU popup, a helicopter icon is displayed on the screen. Use the mouse to position the icon at the location on the contour map where the helicopter is to be added and click the right mouse button to fix the position. A message is then displayed in the PANELWIN graphics window to enter the name of the CAST format file which contains the flight data for the aircraft. After the filename has been entered, the MAIN MENU popup is again displayed. Additional aircraft can now be added.

E. DELETING AIRCRAFT FROM THE SYSTEM

Aircraft can be removed from the CAST system at any time by using the MAIN MENU popup. Individual aircraft can be deleted or all aircraft can be deleted at once. The DELETE A PLATFORM option of the MAIN MENU popup is used in CAST in the same mannar as it is used in MPS. When an aircraft is deleted from the CAST system, the associated flight data file is also deleted.

F. SELECTING A PLATFORM TO OPERATE

When all aircraft have been added to the CAST system, an aircraft must be selected for operation. The flight data of the aircraft selected for operation will be utilized for the display of the graphics in the PANELWIN and DISPLAYWIN graphics windows. The pilot's and copilot's views will be displayed from the aircraft selected for operation. If an external eyepoint is selected, the viewpoint will be the center of gravity of the aircraft under operation. The SELECT A PLATFORM TO OPERATE option of the MAIN MENU popup is used to select an aircraft as is done in MPS. Once an aircraft is selected for operation, the three dimensional graphics of the aircraft flight are displayed. The aircraft being operated can be changed at any time by returning to the MAIN MENU popup. A RETURN TO MAIN MENU option is provided in the FLYING PLATFORM OPERATING MENU popup so that this can be done.

G. CHANGING THE EYEPOINT

When the three dimensional depiction of the aircraft flight is being displayed, the eyepoint can be changed. The EYE POSITION OPTIONS roll-off-the-side-menu is selected from the FLYING PLATFORM OPERATING MENU popup to do this. The EYE POSITION OPTIONS menu allows selection of the eyepoint at the pilot station (PILOT SEAT option), copilot station (COPILOT SEAT option), or outside the aircraft (OUTSIDE HELICOPTER option). When the pilot or copilot station is selected, DIAL4 and DIAL5 of the dial box are used to control the side to side and up and down motion for the look position. When an outside eyepoint is selected;

DIAL0, DIAL2, and DIAL3 are used to control the position of the eyepoint. DIAL1 is used to vary the field of view regardless of the eyepoint.

H. SELECTING AN ALTITUDE OPTION

The ALTITUDE OPTIONS roll-off-the-side menu is selected from the FLYING PLATFORM OPERATING MENU popup to change the method for which the altitude of the aircraft under operation is computed. The ALTITUDE OPTIONS menu allows the aircraft to be displayed at a constant altitude of 200 meters above ground level (CONSTANT ALTITUDE option), at the actual altitudes as recorded in the flight data (ACTUAL ALTITUDE option), or at altitudes which are computed by smoothing the actual flight altitude values (SMOOTHED ALTITUDE option).

I. SELECTING A FRAME DELAY OPTION

The rate at which the graphics frames are displayed can be varied with the FRAME DELAY OPTIONS roll-off-the-side menu. This menu can be selected from either the FLYING PLATFORM OPERATING MENU popup or the MAIN MENU popup. A one, two, or five second delay between display of frames can be selected with the ONE SECOND DELAY, TWO SECOND DELAY, or FIVE SECOND DELAY options. A manual, single step forward/backward option can be selected with the MANUAL FORWARD/BACKWARD option. When this option is selected, the left mouse button is used to step backward one frame at a time and the right mouse button is used to step forward one frame at a time. A frame delay can be eliminated by selecting the NORMAL SPEED - NO DELAY option from the FRAME DELAY OPTIONS menu.

J. EXITING CAST

To exit from the CAST system, the EXIT THE PROGRAM option is used. This option can be selected from the OPENING MENU, AREA SELECT MENU, MAIN MENU, or FLYING PLATFORM OPERATING MENU popups.

APPENDIX B

SELECTED CAST SOURCE CODE

```

/*****
FILENAME      : addveh.c
CALLED BY     : do_the_add
CALLS         : mousescreentoterrain
               setwindow
               do_the_filename
               get_next_alt
MODIFIED      : 7/13/88
PERSON        : David Jennings
MODIFIED      : April 1989
PERSON        : CPT(P) Mark J. Christian
. 'OD         : Modified for the CAST system to read in flight data and smooth altitude data.
PURPOSE       : Allocate storage & initialize structure for new node in vehlist linked list. The
               node is added at the end of the list. Reads in flight data for the aircraft and
               performs altitude smoothing.
*****/

#include "Cast.h"
#include <math.h>

addveh(sx,sy,wx,wy,vehtype)

short sx;      /* x screen coord of vehicle */
short sy;      /* y screen coord of vehicle */
float wx;      /* x world coord of vehicle */
float wy;      /* y world coord of vehicle */
short vehtype; /* vehicle type */
{
    extern Vehicle *vehlist;
    extern Vehicle *vehlistend;
    extern Object vehicon[];
    extern short numveh[];

    Vehicle *temp;
    Flt_data_rec *flt_temp,*flt_temp2,*flt_data_list_end;
    float angle,ax,ay,altchg,smooth[6];
    short win,sample_type;
    int i;
    float tx,ty;
    char *malloc();
    Boolean done = FALSE;
    char filename[80];
    FILE *f;

    /* allocate storage for new node */
    temp = (Vehicle *) (malloc(sizeof(Vehicle)));

```

```

temp->flt_data = NULL; /* set initially to NULL */
temp->next = NULL;

/* READ IN THE FLIGHT RECORDER DATA */

/* Open the flight data file for reading */
f = NULL;
while(f == NULL) /* loop until a valid data file name is entered */
{
    do_the_filename(filename);
    f = fopen(filename,"r");
}

/* read in the aircraft tail number */
fscanf(f,"%d",&temp->tail_nbr);

/* read in the flight data */
sample_type = 0;
while(!done)
{
    flt_temp = (Flt_data_rec *) (malloc(sizeof(Flt_data_rec)));

    switch(sample_type)
    {
        case 0: if(fscanf(f,"%d%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f",
            &flt_temp->frame_no,&flt_temp->time,&flt_temp->pitch,
            &flt_temp->roll,&flt_temp->yaw,&flt_temp->long_cyclic,
            &flt_temp->lat_cyclic,&flt_temp->pedals,
            &flt_temp->collective,&flt_temp->airspeed,
            &flt_temp->altitude,&flt_temp->alt_rate,
            &flt_temp->eng1_torque,&flt_temp->eng2_torque,
            &flt_temp->rotor_rpm,&flt_temp->stab_pos,
            &flt_temp->discrete1,&flt_temp->discrete2) != EOF)
        {
            flt_temp->frequency = TYPE18; /* set sample type */
            flt_temp->next = NULL; /* indicate end of list */
            sample_type++; /* increment sample type */

            /* add the record to the list */
            if(temp->flt_data == NULL)
                temp->flt_data = flt_temp;
            else
                flt_data_list_end->next = flt_temp;

            if(temp->flt_data != flt_temp)
                flt_temp->prev = flt_data_list_end;
            else
                flt_temp->prev = flt_temp;

            flt_data_list_end = flt_temp;
        }
        else
            done = TRUE;
        break;

        case 4: if(fscanf(f,"%d%f%f%f%f%f%f%f%f%f%f%f%f%f",
            &flt_temp->frame_no,&flt_temp->time,&flt_temp->pitch,
            &flt_temp->roll,&flt_temp->yaw,&flt_temp->long_cyclic,
            &flt_temp->lat_cyclic,&flt_temp->pedals,
            &flt_temp->collective,&flt_temp->airspeed,
            &flt_temp->altitude,&flt_temp->alt_rate,

```

```

        &flt_temp->eng1_torque,&flt_temp->eng2_torque,
        &flt_temp->rotor_rpm) != EOF)
    {
        flt_temp->frequency = TYPE15;           /* set sample type */
        flt_temp->next = NULL;                  /* indicate end of list */
        sample_type++;                          /* increment sample type */

        /* add the record to the list */
        if(temp->flt_data == NULL)
            temp->flt_data = flt_temp;
        else
            flt_data_list_end->next = flt_temp;

        if(temp->flt_data != flt_temp)
            flt_temp->prev = flt_data_list_end;
        else
            flt_temp->prev = flt_temp;

        flt_data_list_end = flt_temp;
    }
    else
        done = TRUE;
    break;

    case 2:
case 6: if(fscanf(f,"%d%f%f%f%f%f%f%f",
        &flt_temp->frame_no,&flt_temp->time,&flt_temp->pitch,
        &flt_temp->roll,&flt_temp->yaw,&flt_temp->long_cyclic,
        &flt_temp->lat_cyclic,&flt_temp->pedals,
        &flt_temp->collective) != EOF)
    {
        flt_temp->frequency = TYPE9;           /* set sample type */
        flt_temp->next = NULL;                  /* indicate end of list */
        sample_type++;                          /* increment sample type */

        /* add the record to the list */
        if(temp->flt_data == NULL)
            temp->flt_data = flt_temp;
        else
            flt_data_list_end->next = flt_temp;

        if(temp->flt_data != flt_temp)
            flt_temp->prev = flt_data_list_end;
        else
            flt_temp->prev = flt_temp;

        flt_data_list_end = flt_temp;
    }
    else
        done = TRUE;
    break;

case 1:
case 3:
case 5: if(fscanf(f,"%d%f%f%f",
        &flt_temp->frame_no,&flt_temp->time,&flt_temp->pitch,
        &flt_temp->roll,&flt_temp->yaw) != EOF)
    {
        flt_temp->frequency = TYPE5;           /* set sample type */
        flt_temp->next = NULL;                  /* indicate end of list */
        sample_type++;                          /* increment sample type */

```

```

        /* add the record to the list */
        if(temp->flt_data == NULL)
            temp->flt_data = flt_temp;
        else
            flt_data_list_end->next = flt_temp;

        if(temp->flt_data != flt_temp)
            flt_temp->prev = flt_data_list_end;
        else
            flt_temp->prev = flt_temp;

        flt_data_list_end = flt_temp;
    }
    else
        done = TRUE;
    break;

case 7: if(fscanf(f,"%d%f%f%f%f",
    &flt_temp->frame_no,&flt_temp->time,&flt_temp->pitch,
    &flt_temp->roll,&flt_temp->yaw) != EOF)
    {
        flt_temp->frequency = TYPE5;        /* set sample type */
        flt_temp->next = NULL;              /* indicate end of list */
        sample_type = 0;                   /* increment sample type */

        /* add the record to the list */
        if(temp->flt_data == NULL)
            temp->flt_data = flt_temp;
        else
            flt_data_list_end->next = flt_temp;

        if(temp->flt_data != flt_temp)
            flt_temp->prev = flt_data_list_end;
        else
            flt_temp->prev = flt_temp;

        flt_data_list_end = flt_temp;
    }
    else
        done = TRUE;
    break;
}

/* Close the flight data file */
fclose(f);

temp->curr_flt_data_rec = NULL;
temp->end_of_flt_data = FALSE;

/* Perform the altitude smoothing. TYPE18 and TYPE15 data */
/* records are first used. These records contain flight */
/* recorder data for altitude. To smooth the altitudes, */
/* the flight data recorder altitude is multiplied by two */
/* and added to the two previous and the two next flight */
/* data recorder altitudes. This sum is then divided by */
/* six to arrive at the smoothed altitude. */

flt_temp = temp->flt_data;
smooth[0] = (float)flt_temp->altitude;

```

```

smooth[1] = smooth[0];
smooth[2] = smooth[0];
flt_temp2 = flt_temp;
flt_temp = get_next_alt(flt_temp);
smooth[3] = (float)flt_temp->altitude;
flt_temp = get_next_alt(flt_temp);
smooth[4] = (float)flt_temp->altitude;
while(flt_temp2 != flt_temp)
{
    flt_temp2->s_alt =
        (smooth[0]+smooth[1]+(2*smooth[2])+smooth[3]+smooth[4]) / 6;
    flt_temp2 = get_next_alt(flt_temp2);
    flt_temp = get_next_alt(flt_temp);
    smooth[5] = (float)flt_temp->altitude;
    for(i = 0; i < 5; i++)
        smooth[i] = smooth[i+1];
}
flt_temp2->s_alt =
    (smooth[0]+smooth[1]+(2*smooth[2])+smooth[3]+smooth[4]) / 6;

/* Now smooth the TYPE5 and TYPE9 data records. These are the */
/* record types which lie between the records that contain */
/* flight data recorder altitudes. Three records are between */
/* each "pair" of flight data altitude records. The smoothed */
/* altitudes in these records are computed as follows. The */
/* smoothed altitudes immediately surrounding the TYPE5 and */
/* TYPE9 record set are found. The difference is computed */
/* between these two smoothed altitudes. One fourth of this */
/* difference is then added successively to the three */
/* intermediate records to arrive at their altitudes. */

flt_temp = temp->flt_data;
flt_temp2 = flt_temp;
flt_temp = get_next_alt(flt_temp);
while(flt_temp2 != flt_temp)
{
    altchg = (flt_temp->s_alt - flt_temp2->s_alt) / 4.0;
    for(i = 0; i < 3; i++)
    {
        flt_temp2 = flt_temp2->next;
        flt_temp2->s_alt = flt_temp2->prev->s_alt + altchg;
    }
    flt_temp = get_next_alt(flt_temp);
    flt_temp2 = flt_temp2->next;
}
while(flt_temp2->next != NULL)
{
    flt_temp2 = flt_temp2->next;
    flt_temp2->s_alt = flt_temp->s_alt;
}

/* Compute terrain coords of screen coords */
mousescreentoterrain(sx,sy,&tx,&ty,&win);

/* Insure that we are in the correct window. */
setwindow(MAPWIN);

/* convert angle from x axis in radians */
temp->cse = temp->flt_data->yaw;
angle = (temp->cse <= 90) ? (90.0 - temp->cse) * DTOR

```

```

        : (450.0 - temp->cse) * DTOR;

/* determine world coordinates for vehicle course arrowhead */
ax = wx + ARROW_LENGTH * (float)(cos((double)angle));
ay = wy + ARROW_LENGTH * (float)(sin((double)angle));

/* now draw the icon and arrow indicating vehicle's course */
frontbuffer(TRUE);
    move2(wx,wy);
    callobj(vehicon[vehtype]);
frontbuffer(FALSE);

frontbuffer(TRUE);
    linewidth(2);
    setcolor(WHITE);
    move2(wx,wy);
    draw2(ax,ay);
    draw2(ax - ARROW_WING_LENGTH *
        (float)(cos((double)(angle + ARROW_WING_ANGLE*DTOR))),
        ay - ARROW_WING_LENGTH *
        (float)(sin((double)(angle + ARROW_WING_ANGLE*DTOR))));
    move2(ax,ay);
    draw2(ax - ARROW_WING_LENGTH *
        (float)(cos((double)(angle - ARROW_WING_ANGLE*DTOR))),
        ay - ARROW_WING_LENGTH *
        (float)(sin((double)(angle - ARROW_WING_ANGLE*DTOR))));
    linewidth(1);
frontbuffer(FALSE);

/* fill in structure members */

temp->t = vehtype;
temp->x = tx;
temp->y = gnd_level(tx,ty);
temp->z = -ty;

temp->ang = angle;
    temp->roll = temp->flt_data->roll;
    temp->pitch = temp->flt_data->pitch;

    temp->lookaz = 0.0;
    temp->lookel = 0.0;

temp->gridx = (short)(tx / TENTH KM);
temp->gridz = (short)(ty / TENTH KM);
temp->alt = temp->y + COBRA_INIT_HT;
    temp->vel = 0.0;
temp->sx = (float)(sx);
temp->sy = (float)(sy);

/* now add the vehicle node to the vehlist linked list. */
if (vehlist == NULL)
    vehlist = temp;
else
    vehlistend->next = temp;

vehlistend = temp;

/* now there is one more vehicle of this type. */
numveh[vehtype] += 1;
}

```

```

/*****
FILENAME      : calc_look_parameters.c
CALLED BY     : set_driven_view
CALLS         : transform_body_to_world
               update_look_pos_cobra
CREATED       :
PERSON        : MAJ William Teter
MODIFIED      : 16 May 89
PERSON        : CPT(P) Mark J. Christian
MOD           : Modified to conform to the CAST program
PURPOSE       : Calculates parameters for view from helicopter
*****/

```

```
#include "Cast.h"
```

```
calc_look_parameters(offset_x,offset_y,offset_z,eye_x,eye_y,eye_z,
                    pt_x,pt_y,pt_z)
```

```
float  offset_x,offset_y,offset_z;
float  *eye_x,*eye_y,*eye_z;
float  *pt_x,*pt_y,*pt_z;
```

```

{
    extern  Vehicle *driven;
    extern  short   eye_position;

    float   viewposn_offset_x, viewposn_offset_y, viewposn_offset_z;

    /* Calculate eye position offset from center of platform in world
       coordinates. */
    transform_body_to_world(driven->ang,
                           driven->pitch * DTOR,driven->roll * DTOR,
                           offset_x, offset_y, offset_z,
                           &viewposn_offset_x,&viewposn_offset_y,
                           &viewposn_offset_z);

    /* Now combine platform position and eye offset to get world coordinates
       of eye position. */
    *eye_x = driven->x + viewposn_offset_x;
    *eye_y = driven->y + viewposn_offset_y;
    *eye_z = driven->z + viewposn_offset_z;

    /* Calculate point looked at for view */
    if(eye_position != OUTSIDE)
        update_look_pos_cobra(*eye_x,*eye_y,*eye_z,driven->ang,
                              driven->pitch * DTOR,driven->roll * DTOR,
                              -driven->lookaz * DTOR,-driven->lookel * DTOR,
                              pt_x,pt_y,pt_z );
    else
    {
        *pt_x = driven->x;
        *pt_y = driven->y;
        *pt_z = driven->z;
    }
}

```

```

/*****
FILENAME      : convert.c
CALLED BY     : NONE
CALLS         : read_field
               read_field2
CREATED       : 6 April 1989
PERSON        : CPT(P) Mark J. Christian
PURPOSE       : This program reads a flight data file which is in a Lotus
               1-2-3 format and reformats it to the CAST format. Flight
               data is recorded in one second cycles with eight intervals
               per cycle. Flight parameters are recorded at varying sample
               rates (1, 2, 4, or 8 samples per second). The specific
               interval within a cycle determines the parameters which are
               recorded. A cycle begins with a sample which contains all
               flight parameters.
*****/

```

```

#include <stdio.h>
#include "fltdata.h"

```

```

main()
{
    char chr150[150],fr[7],ct[10],pa[7],ra[7],ya[6],longs[6];
    char lats[6],pp[6],cs[6],as[6],alt[6],altr[6],et1[6],et2[6],rotor[6];
    char stab[6],chr;
    unsigned int discrete1,discrete2;
    int i,j;
    FILE *f,*g;

    /*
       open the Lotus 1-2-3 format flight data file, the filename in the
       open command must match the file to be converted, therefore editing
       may be needed here
    */
    f = fopen("ac506z.tm","r");

    /*
       open a temporary file for writing the CAST format file into, this
       filename may be changed as needed
    */
    g = fopen("tempfile.z","w");

    i = 0; /* index for input string */
    j = 0; /* cycle interval counter */

    /*
       scan the entire input file until End of File is encountered
    */
    while(fscanf(f,"%c",&chr) != EOF)
    {
        /*
           read in characters into the input string until a line feed
           is encountered
        */
        if (chr != '\012')
        {
            chr150[i] = chr;
            i++;
        }
    }
    /*

```



```

when a line feed is encountered, process the data read in
*/
else
{
    switch(j)
    {
        /*
        five parameters are processed for intervals
        1, 3, and 5 and the interval counter is incremented
        */
        case 1:
        case 3:
        case 5:
            read_field(FR,fr,chr150);
            read_field(CT,ct,chr150);
            read_field(PA,pa,chr150);
            read_field(RA,ra,chr150);
            read_field(YA,ya,chr150);
            j++;
            /*
            write out the parameters in CAST format
            */
            fprintf(g,"%s %s %s %s %s\n",fr,ct,pa,ra,ya);
            break;

        /*
        five parameters are processed for interval 7
        and the interval counter is reset to zero
        */
        case 7:
            read_field(FR,fr,chr150);
            read_field(CT,ct,chr150);
            read_field(PA,pa,chr150);
            read_field(RA,ra,chr150);
            read_field(YA,ya,chr150);
            j = 0;
            /*
            write out the parameters in CAST format
            */
            fprintf(g,"%s %s %s %s %s\n",fr,ct,pa,ra,ya);
            break;

        /*
        nine parameters are processed for intervals 2 and
        6 and the interval counter is incremented
        */
        case 2:
        case 6:
            read_field(FR,fr,chr150);
            read_field(CT,ct,chr150);
            read_field(PA,pa,chr150);
            read_field(RA,ra,chr150);
            read_field(YA,ya,chr150);
            read_field(LONGS,longs,chr150);
            read_field(LATS,lats,chr150);
            read_field(PP,pp,chr150);
            read_field(CS,cs,chr150);
            j++;
            /*
            write out the parameters in CAST format

```

```

    */
    fprintf(g, "%s %s %s %s %s %s %s %s %s\n",
        fr, ct, pa, ra, ya, longs, lats, pp, cs);
    break;

/*
    fifteen parameters are processed for interval 4
    and the interval counter is incremented
    */
case 4:
    read_field(FR, fr, chr150);
    read_field(CT, ct, chr150);
    read_field(PA, pa, chr150);
    read_field(RA, ra, chr150);
    read_field(YA, ya, chr150);
    read_field(LONGS, longs, chr150);
    read_field(LATS, lats, chr150);
    read_field(PP, pp, chr150);
    read_field(CS, cs, chr150);
    read_field(AS, as, chr150);
    read_field(ALT, alt, chr150);
    read_field(ALTR, altr, chr150);
    read_field(ET1, et1, chr150);
    read_field(ET2, et2, chr150);
    read_field(ROTOR, rotor, chr150);
    j++;
/*
    write out the parameters in CAST format
    */
    fprintf(g, "%s %s %s %s %s %s %s %s %s %s",
        fr, ct, pa, ra, ya, longs, lats, pp, cs, as, alt);
    fprintf(g, " %s %s %s %s\n",
        altr, et1, et2, rotor);
    break;

/*
    sixteen parameters and fifty discrete values are
    processed for interval 0 and the interval counter
    is incremented
    */
case 0:
    read_field(FR, fr, chr150);
    read_field(CT, ct, chr150);
    read_field(PA, pa, chr150);
    read_field(RA, ra, chr150);
    read_field(YA, ya, chr150);
    read_field(LONGS, longs, chr150);
    read_field(LATS, lats, chr150);
    read_field(PP, pp, chr150);
    read_field(CS, cs, chr150);
    read_field(AS, as, chr150);
    read_field(ALT, alt, chr150);
    read_field(ALTR, altr, chr150);
    read_field(ET1, et1, chr150);
    read_field(ET2, et2, chr150);
    read_field(ROTOR, rotor, chr150);
    read_field(STAB, stab, chr150);
    read_field2(&discrete1, &discrete2, chr150);
    j++;
/*

```

```

        write out the parameters in CAST format
    */
    fprintf(g,"%s %s %s %s %s %s %s %s %s %s",
        fr,ct,pa,ra,ya,longs,lats,pp,cs,alt);
    fprintf(g," %s %s %s %s %s %s %8d %8d\n",
        altr,et1,et2,rotor,stab,discrete1,discrete2);
    break;
    }
    i = 0; /* reset input string index */
}
/*
close the Lotus 1-2-3 and CAST format data files
*/
fclose(f);
fclose(g);
}

/*****
PROCEDURE   : read_field
PURPOSE     : This procedure extracts flight parameter data from the
               input string so that it may be written out in CAST format.
               Each parameter is in a specific position in the input
               string.
*****/

read_field(field_name,field,input_line)

int  field_name;
char *field,*input_line;

{
    int k;

    /*
    the field name (parameter name) determines where in the input
    string to extract the data
    */
    switch(field_name)
    {
        case FR: /* FRame (line) counter */
            for(k = 0;k < 6;k++)
                field[k] = input_line[k];
            field[6] = '\0';
            break;

        case CT: /* Combined Time */
            for(k = 6;k < 15;k++)
                field[k-6] = input_line[k];
            field[9] = '\0';
            break;

        case PA: /* Pitch Attitude */
            for(k = 15;k < 21;k++)
                field[k-15] = input_line[k];
            field[6] = '\0';
            break;

        case RA: /* Roll Attitude */
            for(k = 21;k < 27;k++)

```

```

        field[k-21] = input_line[k];
        field[6] = '\0';
        break;

case YA: /* Yaw Attitude */
    for(k = 27;k < 32;k++)
        field[k-27] = input_line[k];
    field[5] = '\0';
    break;

case LONGS: /* LONGitudinal Stick */
    for(k = 32;k < 37;k++)
        field[k-32] = input_line[k];
    field[5] = '\0';
    break;

case LATS: /* LATitudinal Stick */
    for(k = 37;k < 42;k++)
        field[k-37] = input_line[k];
    field[5] = '\0';
    break;

case PP: /* Pedal Position */
    for(k = 42;k < 47;k++)
        field[k-42] = input_line[k];
    field[5] = '\0';
    break;

case CS: /* Collective Stick */
    for(k = 47;k < 52;k++)
        field[k-47] = input_line[k];
    field[5] = '\0';
    break;

case AS: /* AirSpeed */
    for(k = 52;k < 57;k++)
        field[k-52] = input_line[k];
    field[5] = '\0';
    break;

case ALT: /* ALTitude */
    for(k = 57;k < 62;k++)
        field[k-57] = input_line[k];
    field[5] = '\0';
    break;

case ALTR: /* ALTitude Rate */
    for(k = 62;k < 67;k++)
        field[k-62] = input_line[k];
    field[5] = '\0';
    break;

case ET1: /* Engine Torque #1 */
    for(k = 67;k < 72;k++)
        field[k-67] = input_line[k];
    field[5] = '\0';
    break;

case ET2: /* Engine Torque #2 */

```

```

        for(k = 72;k < 77;k++)
            field[k-72] = input_line[k];
        field[5] = '\0';
        break;

    case ROTOR: /* ROTOR rpm */
        for(k = 77;k < 82;k++)
            field[k-77] = input_line[k];
        field[5] = '\0';
        break;

    case STAB: /* STABilator position */
        for(k = 82;k < 87;k++)
            field[k-82] = input_line[k];
        field[5] = '\0';
        break;
    }
}

```

```

/*****
PROCEDURE : read_field2
PURPOSE   : This procedure reads the fifty discrete parameters and
            codes them into two unsigned integers. The integers are
            used as bit fields to indicate which discretes are on.
*****/

```

```

read_field2(disc1,disc2,in_line)
unsigned int *disc1,*disc2;
char *in_line;

```

```

{
    int j;
    unsigned int temp,powers;
    temp = 0;
    powers = 16777216;
    /*
    code the first 25 discretes
    */
    for(j = 87;j < 112;j++)
    {
        if(in_line[j] == '\061') /* 1 */
            temp = temp + powers;
        powers = powers / 2;
    }
    *disc1 = temp;
    temp = 0;
    powers = 16777216;
    /*
    code the second 25 discretes
    */
    for(j = 112;j < 137;j++)
    {
        if(in_line[j] == '\061') /* 1 */
            temp = temp + powers;
        powers = powers / 2;
    }
    *disc2 = temp;
}

```

```

/*****
FILENAME      : draw_cobra.c
CALLED BY     : draw_terrain
CALLS         : draw_main_rotor, draw_tail_rotor, draw_tail_pipe
CREATED       : 14 Mar 89
MODIFIED:
AUTHOR        : CPT(P) Mark J. Christian
PURPOSE       : Draws the body of a Cobra helicopter
*****/

```

```

#include "Lightcons.h"
#include "gl.h"
#include "Roudat.h"

```

```

draw_cobra()
{
    extern float  pn4gcobra[90][3];
    extern float  pn3gcobra[16][3];
    extern float  pn6gcobra[2][3];
    extern float  pn4irs[8][3];
    extern float  pn8irs[3];
    extern float  pn4cabin[31][3];
    extern float  pn3cabin[2][3];

    extern float  p4gcobra[90][4][3];
    extern float  p3gcobra[16][3][3];
    extern float  p6gcobra[2][6][3];
    extern float  p4irs[8][4][3];
    extern float  p8irs[8][3];
    extern float  p4cabin[31][4][3];
    extern float  p3cabin[2][3][3];

    extern Matrix mainrotoracc[4][4];
    extern Matrix tailrotoracc[4][4];
    int          i;

    /* draw and rotate the tail rotor, tail rotor is defined about origin */
    /* preserve the Cobra drawing matrix */
    pushmatrix();

    /* build the accumulative tail rotor rotation matrix */
    loadmatrix(tailrotoracc);
    rotate(TAILROTORSPEED,'z');
    getmatrix(tailrotoracc);

    /* get rid of the rotation matrix */
    popmatrix();

    /* make a copy of the Cobra drawing matrix */
    pushmatrix();

    /* translate the tailrotor to the vertical tail */
    translate(-8.0909,1.1212,0.0000);

    /* apply the tail rotor rotation */
    mulmatrix(tailrotoracc);

    /* draw the translated and rotated tail rotor */
    draw_tail_rotor();

    /* restore the Cobra drawing matrix */

```

```

popmatrix();

/* draw and rotate the main rotor, main rotor defined about origin */

/* preserve the Cobra drawing matrix */
pushmatrix();

/*build the accumulative main rotor rotation matrix */
loadmatrix(mainrotoracc);
rotate(MAINROTORSPEED,'y');
getmatrix(mainrotoracc);

/* get rid of the rotation matrix */
popmatrix();

/* get a copy of the Cobra drawing matrix */
pushmatrix();

/* translate the main rotor to the top of the Cobra */
translate(0.0000,1.5152,0.0000);

/* apply the rotor rotation */
multmatrix(mainrotoracc);

/* draw the translated and rotated main rotor */
draw_main_rotor();

/* restore the Cobra drawing matrix */
popmatrix();

/* preserve the Cobra drawing matrix */
pushmatrix();

/* translate the tail pipe to the engine area */
translate(-1.9394,0.6061,0.0000);

/* rotate the tail pipe into position */
rotate(600,'z');

/* draw the translated and rotated tail pipe */
draw_tail_pipe();

/* restore the Cobra drawing matrix */
popmatrix();

/* draw the body of the Cobra */

lmbind(MATERIAL,COBRABODY);
for (i=0;i<90;i++) {
    n3f(pn4gcobra[i]);
    bgnpolygon();
    v3f(p4gcobra[i][0]);
    v3f(p4gcobra[i][1]);
    v3f(p4gcobra[i][2]);
    v3f(p4gcobra[i][3]);
    endpolygon();
}

for (i=0;i<16;i++) {
    n3f(pn3gcobra[i]);
    bgnpolygon();
}

```

```

        v3f(p3gcobra[i][0]);
        v3f(p3gcobra[i][1]);
        v3f(p3gcobra[i][2]);
    endpolygon();
}

for (i=0;i<2;i++) {
    n3f(pn6gcobra[i]);
    bgnpolygon();
        v3f(p6gcobra[i][0]);
        v3f(p6gcobra[i][1]);
        v3f(p6gcobra[i][2]);
        v3f(p6gcobra[i][3]);
        v3f(p6gcobra[i][4]);
        v3f(p6gcobra[i][5]);
    endpolygon();
}

lmbind(MATERIAL,IRS);
for (i=0;i<8;i++) {
    n3f(pn4irs[i]);
    bgnpolygon();
        v3f(p4irs[i][0]);
        v3f(p4irs[i][1]);
        v3f(p4irs[i][2]);
        v3f(p4irs[i][3]);
    endpolygon();
}

n3f(pn8irs);
bgnpolygon();
    v3f(p8irs[0]);
    v3f(p8irs[1]);
    v3f(p8irs[2]);
    v3f(p8irs[3]);
    v3f(p8irs[4]);
    v3f(p8irs[5]);
    v3f(p8irs[6]);
    v3f(p8irs[7]);
endpolygon();

lmbind(MATERIAL,COBRACABIN);
for (i=0;i<31;i++) {
    n3f(pn4cabin[i]);
    bgnpolygon();
        v3f(p4cabin[i][0]);
        v3f(p4cabin[i][1]);
        v3f(p4cabin[i][2]);
        v3f(p4cabin[i][3]);
    endpolygon();
}

for (i=0;i<2;i++) {
    n3f(pn3cabin[i]);
    bgnpolygon();
        v3f(p3cabin[i][0]);
        v3f(p3cabin[i][1]);
        v3f(p3cabin[i][2]);
    endpolygon();
}
}

```



```

/*****
FILENAME      : draw_main_rotor.c
CALLED BY     : draw_cobra
                draw_in_cobra
CALLS         :
CREATED       : 19 Mar 89
MODIFIED      :
AUTHOR        : CPT(P) Mark J. Christian
PURPOSE       : Draws the main rotor of a Cobra helicopter
*****/

```

```

#include "Lightcons.h"
#include "gl.h"

```

```

draw_main_rotor()
{
    extern float pn4bmrotor[8][3];
    extern float pn5bmrotor[4][3];
    extern float pn4gmast[5][3];

    extern float p4bmrotor[8][4][3];
    extern float p5bmrotor[4][5][3];
    extern float p4gmast[5][4][3];

    int i;

    /* draw the Cobra tail rotor */

    glBind(MATERIAL, ROTORSHAFT);
    for (i=0; i<5; i++) {
        n3f(pn4gmast[i]);
        bgnpolygon();
        v3f(p4gmast[i][0]);
        v3f(p4gmast[i][1]);
        v3f(p4gmast[i][2]);
        v3f(p4gmast[i][3]);
        endpolygon();
    }

    glBind(MATERIAL, MROTORBLADE);
    for (i=0; i<8; i++) {
        n3f(pn4bmrotor[i]);
        bgnpolygon();
        v3f(p4bmrotor[i][0]);
        v3f(p4bmrotor[i][1]);
        v3f(p4bmrotor[i][2]);
        v3f(p4bmrotor[i][3]);
        endpolygon();
    }

    for (i=0; i<4; i++) {
        n3f(pn5bmrotor[i]);
        bgnpolygon();
        v3f(p5bmrotor[i][0]);
        v3f(p5bmrotor[i][1]);
        v3f(p5bmrotor[i][2]);
        v3f(p5bmrotor[i][3]);
        v3f(p5bmrotor[i][4]);
        endpolygon();
    }
}

```

```

/*****
FILENAME      :draw_tail_pipe.c
CALLED BY     :draw_cobra
CALLS         :
CREATED       : 20 Mar 89
MODIFIED      :
AUTHOR        : CPT(P) Mark J. Christian
PURPOSE       : Draws the tail pipe of a Cobra helicopter
*****/

```

```

#include "Lightcons.h"
#include "gl.h"

```

```

draw_tail_pipe()
{
    extern float   pn4btp[8][3];
    extern float   pn8btp[3];

    extern float   p4btp[8][4][3];
    extern float   p8btp[8][3];

    int            i;

    /* draw the Cobra tail pipe */

    glBind(MATERIAL, TAILPIPE);
    for (i=0; i<8; i++) {
        n3f(pn4btp[i]);
        bgnpolygon();
        v3f(p4btp[i][0]);
        v3f(p4btp[i][1]);
        v3f(p4btp[i][2]);
        v3f(p4btp[i][3]);
        endpolygon();
    }

    n3f(pn8btp);
    bgnpolygon();
    v3f(p8btp[0]);
    v3f(p8btp[1]);
    v3f(p8btp[2]);
    v3f(p8btp[3]);
    v3f(p8btp[4]);
    v3f(p8btp[5]);
    v3f(p8btp[6]);
    v3f(p8btp[7]);
    endpolygon();
}

```

```

/*****
FILENAME      :draw_tail_rotor.c
CALLED BY     :draw_cobra
CALLS         :
CREATED       : 19 Mar 89
MODIFIED      :
AUTHOR        : CPT(P) Mark J. Christian
PURPOSE       : Draws the tail rotor of a Cobra helicopter
*****/

```

```

#include "Lightcons.h"
#include "gl.h"

draw_tail_rotor()
{
    extern float pn4gshaft[5][3];
    extern float pn7btrotor[4][3];

    extern float p4gshaft[5][4][3];
    extern float p7btrotor[4][7][3];

    int i;

    /* draw the Cobra tail rotor */

    lmbind(MATERIAL, ROTORSHAFT);
    for (i=0; i<5; i++) {
        n3f(pn4gshaft[i]);
        bgnpolygon();
        v3f(p4gshaft[i][0]);
        v3f(p4gshaft[i][1]);
        v3f(p4gshaft[i][2]);
        v3f(p4gshaft[i][3]);
        endpolygon();
    }

    lmbind(MATERIAL, TROTORBLADE);
    for (i=0; i<4; i++) {
        n3f(pn7btrotor[i]);
        bgnpolygon();
        v3f(p7btrotor[i][0]);
        v3f(p7btrotor[i][1]);
        v3f(p7btrotor[i][2]);
        v3f(p7btrotor[i][3]);
        v3f(p7btrotor[i][4]);
        v3f(p7btrotor[i][5]);
        v3f(p7btrotor[i][6]);
        endpolygon();
    }
}

/*****
FILENAME      : get_next_alt.c
CALLED BY     : addveh
CALLS         : NONE
CREATED       : 20 May 1989
PERSON        : CPT(P) Mark J. Christian
MODIFIED      :
PERSON        :
PURPOSE       : Searches flight data records to find those that contain
                  flight recorder altitude data.
*****/

#include "Cast.h"

Flt_data_rec  *get_next_alt(rec_ptr)

```

```

Flt_data_rec    *rec_ptr;

{
    Flt_data_rec *save_ptr;
    Boolean    done;

    save_ptr = rec_ptr;
    done = FALSE;

    while((rec_ptr->next != NULL) && (!done))
    {
        rec_ptr = rec_ptr->next;
        if((rec_ptr->frequency == TYPE15) || (rec_ptr->frequency == TYPE18))
            done = TRUE;
    }

    if(!done)
        return(save_ptr);
    else
        return(rec_ptr);
}

```

```

/*****
FILENAME      : set_driven_view.c
CALLED BY     : drawterrain.c
CALLS         : calc_look_parameters
CREATED       :
PERSON        : MAJ William Teter
MODIFIED      : 15 May 89 (modified for aerial platforms with roll, pitch, & yaw)
PERSON        : CPT(P) Mark J. Christian
PURPOSE       : Sets viewing projection and transformation for driven platform
*****/

```

```

#include "Cast.h"
#include <math.h>

```

```

set_driven_view()

```

```

{
    extern float  x_disp_of_eye[NUMVEHTYPES][3];
    extern float  y_disp_of_eye[NUMVEHTYPES][3];
    extern float  z_disp_of_eye[NUMVEHTYPES][3];
    extern Vehicle *driven;
    extern short  eye_position;
    extern short  fov;

    Coord  local_px,local_py,local_pz;
    Coord  eye_x,eye_y,eye_z;
    float  viewaz,viewelev,viewroll,viewr;
    float  look_offset_angle;

    calc_look_parameters(x_disp_of_eye[driven->t][eye_position],
        y_disp_of_eye[driven->t][eye_position],
        z_disp_of_eye[driven->t][eye_position],
        &eye_x,&eye_y,&eye_z,&local_px,&local_py,&local_pz);

    perspective(fov,1.0,0.1,MAXLOOKDISTF);

    /* calculate twist angle as combination of pitch and roll, i.e.

```

```

        twist is eye effective roll                */

if(eye_position != OUTSIDE)
    viewroll =
        ((float)cos((double)(-driven->lookaz * DTOR)) * driven->roll * DTOR) +
        ((float)sin((double)(-driven->lookaz * DTOR)) * driven->pitch * DTOR);else
    viewroll = 0.0;

lookat(eye_x,eye_y,eye_z,local_px,local_py,local_pz,
        (Angle)-(viewroll * RTOD * 10.0));

}

/*****
FILENAME      : transform_body_to_world.c
CALLED BY     : calc_look_parameters
               update_look_pos_cobra
CALLS         : loadunit
CREATED       :
PERSON        : MAJ William Teter
MODIFIED      : 16 May 89 (modified for CAST program)
PERSON        : CPT(P) Mark J. Christian
PURPOSE       : Transforms coordinate in body axis to world coordinates.
               Uses Iris matrix multiply microcode to avoid sins and cosines.
*****/

#include "Cast.h"
#include <math.h>

transform_body_to_world(azimuth,elevation,roll,dx,dy,dz,eye_x,eye_y,eye_z)
float  azimuth,elevation,roll;
float  dx,dy,dz;
float  *eye_x,*eye_y,*eye_z;
{
    Matrix  offset_mx;

    pushmatrix();
    loadunit(); /* Load unit matrix */

    /* Assumes platform's nose points along positive X axis */
    /* P(world) = P(body) * ROT(azimuth) * ROT(elevation) * ROT(roll) */

    /* Do rotations in reverse gimbal order */
    rotate( (Angle)(azimuth * RTOD * 10), 'Y' );
    rotate( (Angle)(elevation * RTOD * 10), 'Z' );
    rotate( (Angle)(roll * RTOD * 10), 'X' );

    getmatrix(offset_mx); /* Get accumulated rotation matrix */

    /* Pre-multiply rotation matrix by offset vector to get
       world coordinates. */
    *eye_x = dx * offset_mx[0][0] + dy * offset_mx[1][0] +
             dz * offset_mx[2][0];
    *eye_y = dx * offset_mx[0][1] + dy * offset_mx[1][1] +
             dz * offset_mx[2][1];
    *eye_z = dx * offset_mx[0][2] + dy * offset_mx[1][2] +
             dz * offset_mx[2][2];
    popmatrix();
}

```

```

/*****
FILENAME   : update_look_pos_cobra.c
CALLED BY  : calc_look_parameters
CALLS      : transform_body_to_world
MODIFIED   : 7/28/88
PERSON     : David Jennings
MODIFIED   : 16 May 89 (modified for CAST using version in APS by MAJ Teter)
PERSON     : CPT(P) Mark J. Christian
PURPOSE    : Determine flyer's look position (px,py,pz).
*****/

```

```

#include "Cast.h"
#include <math.h>

```

```

update_look_pos_cobra(eye_x,eye_y,eye_z,azimuth,elevation,roll,
                      viewaz,viewelev,px,py,pz)

```

```

float  eye_x,eye_y,eye_z,azimuth,elevation,roll,viewaz,viewelev;
float  *px,*py,*pz;

```

```

{
    float  wdx,wdy,wdz; /* offset in world coordinates */
    float  body_offset_y = (float)(MAXLOOKDISTF * sin((double)viewelev));
    float  distance = (float)(MAXLOOKDISTF * cos((double)viewelev));
    float  body_offset_x = (float)(distance * (float)cos((double)viewaz));
    float  body_offset_z = -(float)(distance * (float)sin((double)viewaz));

    transform_body_to_world(azimuth,elevation,roll,body_offset_x,
                          body_offset_y,body_offset_z,&wdx,&wdy,&wdz);

    *px = eye_x + wdx;
    *py = eye_y + wdy;
    *pz = eye_z + wdz;
}

```

LIST OF REFERENCES

1. Army Regulation 385-40, *Accident Reporting and Records*, 1 April 1987
2. "MY, how time flies when you're making progress", *United States Army Aviation Digest*, February 1989
3. Department of the Army Pamphlet 385-95, *Aircraft Accident Investigation and Reporting*, 15 June 1983
4. Army Regulation 385-95, *Army Aviation Accident Prevention*, 15 November 1982
5. "Flight data recorders are paying off", *United States Army Aviation Digest*, February 1989
6. United States Army Safety Center, *Flight Data Recorder Parameter List*, 22 June 1988
7. Fichten, Mark A., and Jennings, David H., *Meaningful Real-Time Graphics Workstation Performance Measurements*, Master's Thesis, Naval Postgraduate School, Monterey, California, November 1988
8. United States Army Safety Center, *Transcription Program - Output Format*, 3 March 1988
9. Field Manual 1-5, *Instrument Flying and Navigation for Army Aviators*, 15 December 1984
10. Smith, Douglas B., and Streyle, Dale G., *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System*, Master's Thesis, Naval Postgraduate School, Monterey, California, July 1987
11. Oliver, Michael R., and Stahl, David J., *Interactive, Networked, Moving Platform Simulators*, Master's Thesis, Naval Postgraduate School, Monterey, California, February 1988
12. Shannon, Larry R., and Teter, William A., *APS, An Autonomous Platform Simulator*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1989
13. Strong, Randolph P., and Winn, Michael C., *The Moving Platform Simulator II: A Networked Real-Time Visual Simulator with Distributed Processing and Line-of-Sight Displays*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1989

14. Munson, Steven A., *Integrated Support for Manipulation and Display of 3D Objects for the Command and Control Workstation of the Future*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1989

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142
Naval Postgraduate School
Monterey, California 93943-5002 | 2 |
| 3. | Dr. Michael J. Zyda
Naval Postgraduate School
Code 52, Department of Computer Science
Monterey, California 93943-5100 | 3 |
| 4. | CPT Mark J. Christian
107 North 35th Street
Clear Lake, Iowa 50428 | 3 |
| 5. | Commander
United States Army Safety Center
Attention: CSSC-SE
Fort Rucker, Alabama 36362-5363 | 3 |
| 6. | Mr. Mike Tedeschi
United States Army Test and Experimentation Command
Attention: ATCT-TE-TM
Fort Ord, California 93941-7000 | 1 |